DESIGN AND PROTOTYPE OF THE AFIT VIRTUAL EMERGENCY ROOM: A DISTRIBUTED VIRTUAL ENVIRONMENT FOR EMERGENCY MEDICAL SIMULATION

THESIS

Brian W. Garcia, Captain, USAF

AFIT/GCS/ENG/96D-07

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DTIC QUALITY INSPECTED 1

DESIGN AND PROTOTYPE OF THE AFIT VIRTUAL EMERGENCY ROOM: A DISTRIBUTED VIRTUAL ENVIRONMENT FOR EMERGENCY MEDICAL SIMULATION

THESIS

Brian W. Garcia, Captain, USAF

AFIT/GCS/ENG/96D-07

19970210 035

Approved for public release, distribution unlimited

AFIT/GCS/ENG/96D-07

DESIGN AND PROTOTYPE OF THE AFIT VIRTUAL EMERGENCY ROOM: A DISTRIBUTED VIRTUAL ENVIRONMENT FOR EMERGENCY MEDICAL SIMULATION

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Brian W. Garcia, B.S.C.S, M.S.C.I.S.

Captain, USAF

December, 1996

Approved for public release, distribution unlimited

Preface

This thesis would not have been possible without the advice and support of many others. I have relied heavily on the thoughts and ideas of my advisor, Lt Col Martin Stytz, who has made quite a name for himself in the VR community. He gave me the flexibility I needed to steer the project in my own way, and yet provided insight and ingenuity crucial to its successful completion. I also appreciate the time and attention contributed by my committee readers, Major Keith Shomper and Major Sheila Banks. Each provided ideas, asked interesting questions, and made time for several work-in-progress demos.

Special thanks to Dr. Gayl M. Godsell-Stytz, who provided most of the medical domain knowledge I needed to get this project off the ground, and who sacrificed several days of precious off-duty time to somehow get me into several Dayton-area emergency rooms. Thanks also go to Major Cynthia Semones, the Nurse Manager of the Wright-Patterson AFB Emergency Department, and to Dr. Robert Uptmore at the St. Elizabeth's (now Franciscan) Emergency Department for being good sports about my "odd visits" to their Emergency Rooms with a loaded camera.

Additional thanks to Steve Sheasby, who is responsible for the Medical Manager Software upon which this project so heavily depends. Your responsiveness in modifying the Manager for MediGram changes, and occasional suggestions about on how best to organize the MediGrams proved to be quite beneficial. Thanks "Contractor Steve."

I must certainly thank my classmates in the GCS-96D and GCE-96D programs. As a group, your helped ease the agony of many grueling research days and nights. In particular, thanks and appreciation go to my other fellow "night shift" and "weekend shift" lab-mates. Your spirited company (and 72.3 gallons of diet "cheap soda") made enduring the agonizingly cold (and hot) Graphics Lab much, much easier.

Last, but most important, I must thank my lovely wife Claudia. The last 18 months have been as hard on her as they've been on me. But no matter how difficult or stressful the situation, she was always there to help. At long last, it's time to get out of school and back into life. I love you, "chick."

Brian W. Garcia

Table of Contents

List of Figures

List of Tables

Abstract

Due to the increasing complexity of emergency medical care, medical staffs require increasingly sophisticated training systems. Virtual environments offer a low cost means to achieve a widely usable yet sophisticated training capability. The Defense Advanced Research Projects Agency (DARPA) has sponsored the Virtual Emergency Room (VER) project to develop a simulation system that enables emergency department personnel within level I and II emergency rooms to practice emergency medical procedures and protocols. The VER is a simulation facility that uses a distributed virtual environment architecture to enable real-time, multi-participant simulations. The potential advantages of this system include the ability to evaluate and refine treatment skills, and the ability to provide scenario-specific training for mobile military field hospital teams. These advantages will ultimately improve the readiness of emergency department staffs for a wide variety of trauma situations. This thesis represents the initial phase of a several-year research effort.

DESIGN AND PROTOTYPE OF THE AFIT VIRTUAL EMERGENCY ROOM: A DISTRIBUTED VIRTUAL ENVIRONMENT FOR EMERGENCY MEDICAL SIMULATION

1. Introduction

1.1 Background

To meet the need to rapidly train emergency response staffs, medical personnel require an environment where they can realistically practice the skills required to triage, diagnose, and provide medical treatment. The typical civilian or military emergency room setting is one in which time-critical triage and diagnosis must be performed rapidly (often due to trauma, myocardial infarction, aneurysms, etc.) based on a continuing assessment of the patient's ever-changing physiological status.

Unfortunately, professional training for emergency physicians has not matured at the same rapid pace as that of pilot, air traffic control, field combat, and other professional training [NILA93; SATA95]. In fact, emergency medical teams such as those in mobile army surgical hospital (MASH) units typically train as pilots did before the first aircraft trainer was developed. Such training revolves around creating a mental picture in words of a situation for the physician and medical staff, and then allowing them to work together to solve the problems indicated by the word picture they receive [GODS95].

The absence of a collaborative training environment often reduces the effectiveness of the training, because realistic emergency medical scenarios are often very difficult to conceptualize. Because of the ensuing difficulty in training, it can take years for medical staff members to acquire broad-based expertise in emergency medical skills. As a result, rapidly augmenting emergency department staffs is currently impractical.

1.2 Research Objective

Physicians require many unique skills. In order to meet the need for a competent surge capability in emergency medical care, emergency response teams require an environment where they can realistically practice the skills required to triage, diagnose, and treat wounds. A

possible solution to this problem is to develop a distributed virtual environment (DVE) capable of supporting a multi-person emergency medical simulation in real time.

The Virtual Emergency Room (VER), under sponsorship of DARPA, is being developed to permit emergency department personnel within level I and II Emergency Departments to practice the procedural application of emergency medical treatments. The current work represents the initial phase of the VER project in which a single doctor trainee may diagnose, assess, and treat a virtual patient in a fully distributed virtual environment. This initial capability will lay the foundation for several years of follow-on research, to include development of a collaborative team-treatment capability.

The primary beneficiaries of this research are the Emergency Departments in both the military and civilian medical sectors. The potential advantages of the VER simulator include the ability to provide faster and more effective trauma management, the ability to evaluate and refine procedural treatment skills, and the ability to provide scenario-specific training for mobile military field hospital teams. The flexibility of simulation provided by the VER will lead to a training platform for emergency doctors and staffs that *exceeds* the capabilities of existing training techniques.

1.3 Thesis Statement

Design an architecture for emergency medical triage and treatment training within a distributed virtual environment, and demonstrate the feasibility of this architecture by implementing a functional prototype.

1.4 Scope

The scope of this thesis is focused on investigating the initial research phase of the VER project. This involves designing and implementing a distributed virtual environment (DVE) framework capable of supporting interactions between a doctor participant and an independent virtual patient. The design framework and associated prototype includes the design and implementation of a doctor station, a virtual patient monitoring and control station, and specialized medical message formats and management software to support communication between the stations.

Rather than attempt to model all of the characteristics of an actual Emergency Room, the problem is limited to implementing an interaction capability between a doctor trainee station and

a virtual patient. Collaborative interaction between multiple doctor participants is not supported. Furthermore, environmental characteristics such as gravity and inter-object collision detection are not supported. Efficient algorithms for accomplishing these tasks may be added as follow-on enhancements. Finally, a model of human physiology is not implemented in this effort. Rather, a capability to integrate such a model, when one is available, is provided.

Several, but not all, apparatus and treatments available in an actual Level I/II ER are integrated into this prototype. Representative apparatus and associated treatment capabilities are implemented to demonstrate the architecture, but not necessarily to demonstrate a complete ER capability. For instance, due to the computational complexity of surgical simulation, hands-on interaction between doctor trainee and patient is not supported. Demonstration of the DVE concepts upon which the VER is designed is limited to procedural, non-surgical and non-invasive treatment interactions. To further minimize the modeling and computational complexity of this effort, deformable geometry such as tubes, hoses, wires, and organs are not implemented. This is due in large part to technical limitations, which are discussed in Chapter 2.

1.5 Research Goals and Assumptions

The goals of this effort include satisfying the following general statements:

- Create an immersive doctor station ER facility, complete with functional apparatus.
- Design and implement a mechanism to simulate the physiological state of a virtual patient.
- Establish a communication capability between the immersive ER facility and the virtual patient.

Because this is the first of a multi-phase effort, this research project is conducted based on several assumptions:

- Accurate human physiological models that determine how various medical treatments affect vital signs are not required to demonstrate an initial VER capability.
- A typical rendition of a Level I/II emergency room is achievable and the results are recognizable by emergency medical staffs, despite variations in facility layouts and emergency medical apparatus between facilities.
- An emergency medical training facility is a useful and extensible training aid.
- Sheasby's DIS-compliant Network Manager software is available and adaptable to the medical simulation domain [SHEA96].

- Information about emergency medical facilities and rudimentary treatment information is available.

1.6 Approach

The research and development approach for this project incorporates elements of the spiral software development model. Initially, general performance requirements are distilled from the problem statement and guidance meetings. From general requirements, detailed requirements are listed. A system-level design, based on these specific requirements, is developed. The design is then implemented as a rapid prototype and iteratively refined until the behavior specified in the requirements is achieved.

The research for this project employs knowledge from four technical areas: distributed- and non-distributed virtual environments research, human computer interface (HCI) design, software engineering, and emergency medical domain knowledge. Information pertinent to the first three categories was obtained from graduate-level AFIT course notes, conference papers and proceedings, professional and scholarly journals, and textbooks. Medical information was initially found in medical library books and training manuals. Supplemental knowledge was made available by Dr. Gayl M. Godsell-Stytz, an emergency physician. In addition, ER configuration data and photographs were collected from site visits to Level I/II emergency rooms in the Dayton, Ohio metropolitan area.

1.7 Overview

This thesis describes the design and implementation of the VER prototype, to provide an initial training capability for emergency medical simulation. Chapter 2 provides a background on Emergency Medical Facilities, personnel and apparatus. In addition, a summary of the current literature with respect to medical virtual environments is presented. Finally, a discussion of key technologies relevant to this project are discussed, to include DIS, the AFIT Common Object Database (CODB) Architecture, and the IRIS Performer library. Chapter 3 provides a requirements analysis and definition for the VER prototype. Chapter 4 discusses the system-level design of the VER prototype. The implementation details of the VER prototype are then discussed in Chapter 5, followed by a description of the functional results in Chapter 6. Finally, conclusions and recommendations for future work are outlined in Chapter 7.

2. Background

2.1 Introduction

To understand the design approach taken for the Virtual Emergency Room, the reader must have a basic understanding of the domain knowledge behind the simulation as well as the current technologies and issues relevant to its development. This chapter provides a brief background discussion of four topics. First, the personnel and apparatus of a typical hospital Emergency Department are described. Second, useful concepts related to virtual environment technologies are enumerated. Third, a survey of the literature is presented, which describes the state of current research of emergency medical virtual environments. The survey also summarizes the limitations common to most medical VEs. Fourth, the Silicon Graphics' IRIS Performer graphics rendering library is discussed. The Performer library provides the software building blocks necessary for development of immersive VE applications. Finally, supporting research required for the VER prototype design is described.

2.2 Emergency Room Organization and Equipment

An *Emergency Department* (ED) is the department of a hospital in which emergency cases are evaluated and treated [AHD85]. Emergency Rooms (ERs), which are maintained within hospital emergency departments, are the hospital facilities that administer critical care to patients who are seriously ill or injured. Emergency department staffs work in emergency rooms to triage, diagnose, and treat cases that are commonly life-or-death situations. Emergency Departments are required in military and civilian settings. The emergency medical services provided by well-trained Emergency Departments are required for individual- and mass-casualty care in times of both peace and war [SHEE92].

2.2.1 ED Certification.

While the lifesaving mission of emergency rooms is common to all emergency departments, not all emergency rooms are the same. Some emergency rooms are better equipped and better prepared to treat a wider variety of cases. The Emergency Medical Services System Act of 1973 requires a categorization of hospital facilities according to their emergency support

capabilities. This categorization ideally indicates the preparedness of each facility to care for critically ill or wounded patients.

Emergency Rooms nationwide are assigned one of three categories by the Joint Commission on Accreditation of Healthcare Organizations. The certification is based on the capability of each ED to rapidly treat major trauma, burns, spinal cord injuries, cardiac emergencies, poisoning, neonatal emergencies, and psychiatric emergencies. A Level I emergency room provides the most sophisticated care, and has a trauma surgeon, anesthesiologist, and medical staff available for multi-specialty care 24 hours a day [SCHW89]. Level II emergency rooms have most of the equipment required to treat the aforementioned emergencies, and have specialty medical staffs on-call rather than on the premises. Level III emergency rooms are the least sophisticated facilities, and offer only a subset of the aforementioned care.

2.2.2 Emergency Department Organization

Emergency Departments have a wide variety of personnel who are skilled in (or are learning) emergency medical techniques. While the structure of Emergency Departments varies with each hospital, a typical ED personnel structure resembles that shown in Figure 2-1.

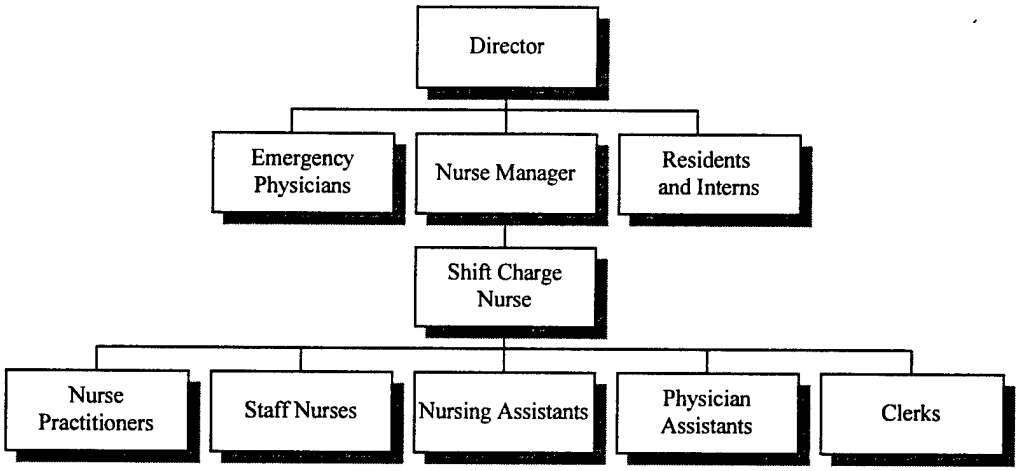


Figure 2-1. Typical ED staff structure [SHEE92; JENK78].

All ED personnel depicted in Figure 2-1, except for management, cooperate to form trauma teams that provide critical care services. The functions of the trauma team participants include the following [SHEE92; JENK78]:

- Staff Nurses and Nurse Practitioners configure equipment, collect vital signs, perform immediate assessments, undress patients, control bleeding, and perform other support functions critical to providing timely emergency care.
- Nursing Assistants perform peripheral duties, and assist nurses as directed.
- Physician Assistants accomplish initial diagnoses, and assist emergency physicians with surgical and diagnostic requirements.
- Clerks document patient details, collect billing information, and maintain contact with caregivers and patient relatives.
- Shift Charge Nurses arrange for specimens, arrange for supplemental blood units, serve as external liaisons, perform administrative coordination, and perform other support tasks as required by physicians.
- Emergency Physicians, Residents and Interns direct the trauma teams, perform patient assessments, provide and direct patient care, perform surgical procedures, and determine diagnostic tests.

In addition to these personnel, Level I emergency rooms also employ orthopedic surgeons, general surgeons, and neurologists available to augment the trauma team when required.

2.2.3 Emergency Room Apparatus.

Just as with emergency department staffing, the type and layout of emergency medical apparatus depends on the hospital. However, most emergency rooms employ a common group of critical care equipment. This apparatus group includes most of the following items [CUMM94; NURS80]:

- Defibrillator/Monitor, used to monitor patient heart rhythm, and provide defibrillation treatment if required. Defibrillation helps reestablish normal contraction rhythms in a heart that is not functioning properly by delivering an electric shock to the heart at different energy levels. Defibrillators are also used to externally pace weak hearts.
- Ventilator/Respirator, used to ventilate the lungs in cases where breathing is obstructed, decreased, or paralyzed. This device moves volumes of air to the lungs at various oxygen levels to simulate the effects of breathing.
- Vital Signs Monitor, provides graphical display of patient vital signs. These devices vary across product lines, but normally provide pulse and blood pressure information.

- Rapid Infusion IV Pump, automatically dispenses IV fluid and blood therapy at controlled flow rates and intervals.
- Pulse Oximeter, used to provide continuous, real time monitoring of patient oxygenation by non-invasively monitoring the patient's finger. Oxygenation monitoring is particularly important when administering anesthesia, which can deprive the body of oxygen and cause hypoxemia (a condition of low blood-oxygen).
- Crash Cart, a movable cart that contains most ER equipment, to include: defibrillator with paddles, electrode jelly, saline pads, CPR board, drugs for cardiac resuscitation, hand-held resuscitators, assorted oral and nasal airways, assorted IV solutions, catheters and needles, arm-boards, tourniquets, arterial blood sampling kit, assorted syringes and needles, scalpels, sterile gauze, tape, swabs, sutures, gloves, and other equipment.
- Patient Gurney, a wheeled stretcher, used to transport patients.
- Utilities, receptacles and pumps capable of providing forced air, oxygen, and suction. These facilities are required for a variety of emergency medical treatments.
- Patient Warmer, a device that provides external surface heating for patients, typically in the form of a thermal blanket or other heat delivery medium.
- Other miscellaneous equipment, such equipment includes directional lighting, biomedical waste "Sharps" container, x-ray back-lighting, shelving, cabinets, and other medical equipment and supplies.

2.3 Virtual Environment Technologies

There are several virtual environment technologies that influenced the design of the VER. This section describes virtual environments and distributed virtual environments. In addition, this section describes a protocol for communication within distributed virtual environment applications known as Distributed Interactive Simulation. Finally, this section concludes with a discussion of the Common Object Database architecture for distributed virtual environment design.

2.3.1 Virtual and Distributed Virtual Environments

Virtual environments (VEs) are synthetic worlds built using Virtual Reality (VR) technology. Although it has been given many definitions, VR may be defined as "a computer-generated technology which allows information to be displayed in a simulated, but life-like

environment [ROSE96]." Virtual environments typically contain several subsystems, such as those shown in Figure 2-2. Most VEs include the dynamic generation of graphics, and recent innovations have introduced the capability for sound and force feedback [GREE96].

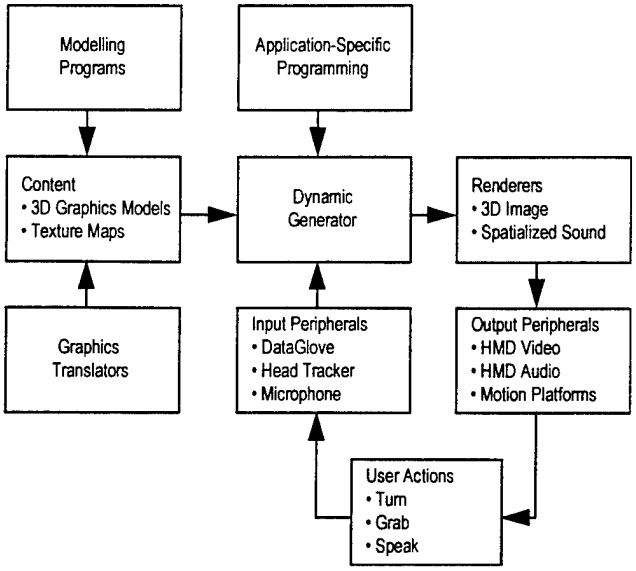


Figure 2-2. Elements of a typical virtual environment [GREE96].

A special type of Virtual Environment, called a Distributed Virtual Environment (DVE), incorporates the technical elements presented in Figure 2-2. However, DVEs add the capability to integrate multiple participants into a loosely coupled simulation framework. Simply stated, a DVE is:

> ...a large-scale, networked, computer-based, virtual world wherein a large number of realistic entities, both human and computer controlled, can interact. Because of the computational scalability afforded by distributed computation, a DVE affords a degree of realism and complexity that can not be achieved in a standalone virtual environment [STYT97].

Thus, DVEs are different in that they rely heavily on communication between various participants. This communication is supported by a commonly understood message protocol that defines the underlying architecture of the simulation.

2.3.2 The DIS Standard

Perhaps the best known standard for DVE communication is the Distributed Interactive Simulation (DIS) Standard. DIS is an IEEE Standard, and a follow-on to the Distributed Simula-

tor Networking (SIMNET) project. The primary purpose of DIS is to create a wide-scale, synthetic virtual environment "through the real-time exchange of data units between distributed, computationally autonomous simulation applications in the form of simulations, simulators, and instrumented equipment interconnected through standard computer communicative services [IST94]." The basic premises upon which the DIS protocol is based include the following [IST94]:

- No central computer controls the entire simulation. Responsibility for simulating the state of each entity rests with individual simulation applications residing in host computers connected via a network.
- Autonomous simulation applications are responsible for maintaining the state of one or more simulation entities. As users operate controls of simulated or actual equipment, the simulation is responsible for sending messages to inform other applications of any and all observable actions.
- A standard protocol is used for communication of simulation events.
- Changes in entity state are communicated by simulation applications.
- Perception of events or other entities is determined by receiving applications.
- Dead reckoning algorithms are used to reduce communications processing. This technique employs extrapolation techniques to limit the rate at which simulations must issue state updates for entities.

The DIS standard specifies 27 Protocol Data Unit (PDU) message formats, which are used to pass information between the entities (human or computer) participating in a shared simulation. The current DIS standard, and the PDU message format repertoire in particular, provide simulation details that describe entity information and interaction, simulation management and performance, radio communications, emissions, and field instrumentation. For a more thorough discussion of the DIS protocol, the reader is referred to the IEEE Standard 1278-1993.

2.3.3 Common Object Database

The software architecture for any DVE must address several types of requirements, to include the number and type of entities in the VE, network architecture and protocols, affordability, deliverability, maintainability, fidelity, rendering frame rate and quality, user

interface, support for automated analysis of the VE, support for decision making, and control of computer-generated actors [STYT97].

There are many existing DVE architectures and paradigms, to include The Cognitive Co-processor Architecture, The MR Toolkit, GIVEN, VERN, DIVE-VR, VEOS Toolkit, and The Veridical User Environment (VUE) [ROSE96b; STYT97]. Unfortunately, there are no standardized architectures that consolidate simulation state information in a widely accessible manner. The absence of this capability led to the design and subsequent implementation of the AFIT Common Object Database (CODB) architecture in the Spring of 1996 [STYT97]. This work couples a distributed intercommunication capability (such as DIS) with a centralized data repository to correctly manage the state of entities in large-scale distributed simulations. The primary advantages of this work with respect to an emergency medical DVE are the availability of specialized simulation support functions and the potential flexibility of using specialized communication messages to drive multi-participant simulations.

The CODB architecture uses a double-buffering mechanism to permit multiple simulation processes to concurrently read and write data stored in a shared data repository. Storage areas are identified by a series of user defined enumerated types. Each storage area is accessed by instantiating a C++ template class that references the storage location, and type-casts it to the appropriate data (or class) type. The flexibility provided in the design allows different data structures to be mixed in the same repository with relative ease. Concurrent access by way of semaphores and shared memory preserves the integrity of data contained within the shared repository. The internal structure of the CODB, to include the double-buffering mechanism, is presented in Figure 2-3.

The effectiveness of the CODB was demonstrated in the AFIT Computer Graphics Laboratory in the Spring of 1996. The demonstration incorporated the CODB architecture as the foundation for a flight simulation application, and was subsequently presented by Adams, Garcia, Zurita, and Wells. The successful results of this demonstration led to the migration of several existing DIS-based projects to the CODB architecture, to include the Virtual Cockpit [ADAM96], The Solar System Modeler [WILL96], and the Synthetic Battle-Bridge [WELL96]. Each of these projects have been successfully ported to the CODB architecture.

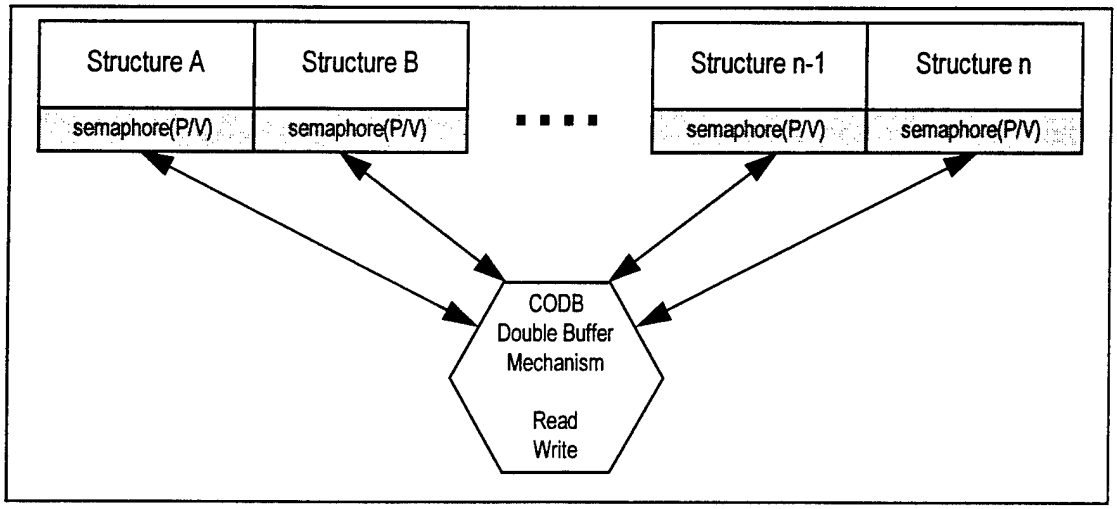


Figure 2-3. Internal structure of Common Object Database (CODB) repository.

2.4 Virtual Environments for Medical Simulation

Recently, a multitude of medical VEs have been developed to simulate the treatment of medical cases. In essence, they represent a radical technology change that will enable physicians to start practicing in a completely digital realm [SATA93a; SATA94; SATA96a]. The interest in medical VE applications is primarily based on the benefits they offer the medical profession as a whole: improved medical training, reduced patient risk, customized curriculums, and fewer animal deaths [SATA93b; MERR94b]. In anticipation of these benefits, many doctors and computer scientists are continually investigating VE technology in a variety of new medical settings [GUPT95; MCGO96; MERR94a; SATA94; SATA96a].

Medical VE systems may be categorized according to five objectives [DUMA93; GREE95; GREE96; SATA94; SATA96a; BREE96]:

1. Assistance before and during medical and surgical procedures
2. Medical education and training
3. Medical database visualization
4. Rehabilitation
5. Rapid design and test of advanced medical apparatus and facilities

Within the realm of education and training, VE technology may be used to provide emergency medical training in civilian and military settings [DUMA93; SATA96a]. However, the literature

does not contain much information of the subject of simulating emergency medicine using VE technology.

The University of Massachusetts at Amherst is developing an intelligent training system that may be used to teach the Advanced Cardiac Life Support (ACLS) protocols, as written by the American Heart Association [ELIO95]. The primary emphasis of this project is to provide elements of an emergency necessary to train a profession to lead cardiac resuscitation teams. The simulator is a non-immersive, single user application developed to explore student centered curriculum theory in an intelligent training system. The graphical presentation and user interface are simplified for training purposes, and consequently do not present a realistic interface into an ER setting.

Another project under investigation is the University of Washington's "Virtual ER" project. This simulator is being developed at University of Washington's Human Interface Technology Laboratory, and is intended "to explore the design space for medical interfaces of the future, in order to determine how immersive augmented space might be used [CGW96]." The project consists of a single user simulator that presents a replica of a local trauma center using still images and textures mapped onto cylindrical geometry. The user participant remains at a fixed position to view the scene, and may only interact with patient data records. Thus, this environment makes extensive use of augmented reality concepts to provide a two-and-a-half dimension environment. Unfortunately, an oversimplification of the interface of this simulator leads to a "look, don't touch" environment, which is not complex enough to accommodate immersive training.

Yet another noteworthy project is an effort to simulate lower extremity battlefield trauma. The research is being conducted by Musculographics Inc., under sponsorship of the Defense Advanced Research Projects Agency (DARPA) Advanced Biomedical Technology Program, and the U.S. Army Medical Research Acquisition Activity [EISL96; KAPL96; SATA96a]. This project will produce a standalone medical VE that simulates surgical care for wounds to lower body extremities. The results of this research are of particular interest in the development of an emergency medical VE, but usable results are not expected until late 1997.

As evidenced by the literature, initial research of emergency medical VEs varies widely in scope and purpose. Existing simulators are either too immature, or are focused on simulating specific treatment protocols. This may be due, in part, to the focus of most current medical VR

literature on the development of highly specialized surgical simulators and remote telepresence medical delivery systems [HON96].

In addition, there has been no published research on the unique requirements of researching a DVE specifically for emergency medical simulation. Only until recently has the medical VR community considered the useful prospects of DVE technology for medical training. Dr. Joseph Rosen, a medical VR researcher at the Dartmouth Medical School, has recently called attention to the need for medical DVE research. He states:

> In order to realize a multiple-player virtual reality system for medical purposes, one needs a system which allows both distribution of the objects in the virtual environment and concurrent interaction of the participants in a real-time fashion, along with high security. Very few existing VR environments meet these requirements [ROSE96b].

Some papers discuss the notion of applying DVE technology to develop medical VR systems that may evolve into tele-medical applications. However, developments on this idea have not been published [ROSE96b].

Despite the lack of prior published research on emergency medical DVEs, the notion of an emergency medical VE is not new. Dr. Andrie Dumay of the TNO physics and electronics laboratory has published several papers that survey the requirements for an emergency medical triage simulator [DUMA93]. The idea of using medical VE technology to host medical training programs for combat medical teams has also been suggested by Dr. Dumay, who suggests that "a programme can be facilitated with computer-based training equipment and training in a Virtual Environment to allow for flexible "on-demand" training focused to a military deployment [sic] [DUMA96]." This vision has recently been expanded to include the following profile for a complete training facility:

> The emergency room of a hospital is a theater that can only function properly when medical staff are well prepared and fully informed on procedures and protocols. This requires specialist training, which may be facilitated with a VE training and simulation system. In such a system, the real emergency room can be modeled, including beds, patient tables, drawers, curtains, surgery facilities, infusion pumps, etc. The drawers may contain bandages, clamps, and syringes. In principle, a virtual patient can be exposed to any injury. In an interactive VE training session, an injury can be treated following a selected protocol, giving the subject the ability to cure the patient or to inflict even worse injuries. In such a training session, the real atmosphere in an emergency-room can be approximated. Even a certain level of stress can be induced to the subject by tightening time constraints and introducing computer-based models of physiology that dynamically adopt to the medical interventions [DUMA96].

Despite the vision, there are no published accounts of such a simulator having been built. However, the basis for an architectural blueprint for a truly distributed emergency medical VE, such as the one just described, is presented by Dr. Godsell-Stytz [GODS95]. The concepts embodied in this vision document include many of the preceding concepts enumerated by Dr. Dumay. In particular, this blueprint expands on the idea of developing an emergency medical VE in which one or more medical participants work to treat a virtual patient in a distributed setting.

2.4.1 Limitations of Existing Medical Virtual Environments

Although the body of knowledge pertaining to medical VE research is relatively small, there have been many attempts to build working prototypes for a variety of medical training purposes. The researchers of early projects, as well as visionaries for future research, have been quick to identify several technical challenges facing medical VE research.

Many medical VE projects have struggled with complex design-related issues. The most prominent of these issues, summarized by Merril, include locating expertise in educational content preparation, conducting anatomic detail planning, preparing training and instructional design, analyzing mechanical engineering requirements, and securing graphics programming expertise [MERR94a]. These issues face all medical VE projects, because each must be addressed in every design. Most projects ultimately cope with these issues, but technical challenges such as interaction, avatar fidelity, and hardware maturity remain.

2.4.1.1 Medical VE Interaction

In general, existing medical VE systems are immature with respect to their capabilities for object interaction. The physical characteristics of cutting, touching, lifting, pushing and pulling are computationally expensive and thus difficult to model in a synthetic environment [SATA96b]. Nevertheless, they are extremely important aspects of most medical procedures. The ability to render realistic patients and correctly model the physical aspects of interacting with them is the focus of much research. Interaction challenges common to most medical VE applications are widely discussed in the literature [COLE94; SATA93b; SATA94; SATA95; SATA96b; THAL94] and summarized in Table 2-1.

Challenge	Description
Image Fidelity	Do the images and objects in the VE appear real?
Object properties	Do objects deform when grabbed and fall when dropped?
Object Interactivity	Do objects interact with other objects in a realistic manner?
Sensory Feedback	Do interfaces provide force feedback and tactile response?
Reactivity	Do objects react to manipulation correctly?

Table 2-1. Research targets for improved medical VE interaction [SATA96a].

Image fidelity is a problem due to requirements for realistic graphics and real-time interactivity. This problem has eased with the development of high performance graphics hardware and software, but is not yet fully resolved. Another problem is the modeling of the various potential object properties within a medical VE. Correct object modeling should permit virtual objects to "behave" correctly, so that they deform when grabbed and fall when dropped. Thus, high fidelity deformation and kinematic models are still required. The physical simulation problem also extends to how objects interact with other objects. Interactivity modeling defines the how objects behave when they come into contact with one another, such as surgical instruments and organs. The problem of limited sensory feedback capabilities impacts how the medical VE approximates the feel of a medical simulation. Sensory feedback includes the requirement for force feedback and tactile response hardware that permits instruments to "feel" like they are cutting tissue, with various resistance to different virtual tissues. Finally, reactivity modeling is required to simulate how objects react to manipulation. For example, bleeding should result from a cut artery and bile leakage should result from a punctured gall bladder.

As stated by Satava, "As powerful as computers are today, they are still an enormous distance away from being powerful enough to compute all the requirements for all five components simultaneously; therefore, there must be tradeoffs [SATA96a]." The efficiency of the image rendering and of kinematic object models are rapidly improving, but efficient solutions to the remaining challenges are not yet available [SATA94].

2.4.1.2 Patient Avatar Fidelity

A patient avatar is a representation of an actual patient within a medical VE. For any medical VE to be deemed "mature," a realistic patient avatar must be included in the simulation [SATA96a]. Simply stated, the quality of the patient avatar imposes an upper bound on the degree of realism for any medical simulator. Accurate patient avatars are so important that Satava suggests a taxonomy to classify the maturity of medical VEs with respect to the fidelity

of the patient avatar [SATA96a]. This taxonomy, which may be directly applied to most medical VE simulation systems, is summarized in Table 2-2.

Medical VE Generation	Patient Avatar Properties	Distinguishing Characteristics
1	Geometric anatomy	3-D physical shapes
2	Physical dynamics modeling	kinematics, deformations
3	Physiologic properties	bleeding, leaking
4	Microscopic anatomy	neurovascular, glandular
5	Biochemical systems	endocrine, immune, shock

Table 2-2. Medical VE taxonomy based on robustness of patient avatar [SATA96a].

Creating 3-D virtual humans has proven to be one of the biggest challenges of medical VE research thus far. The extreme complexity of the human body, and the intricacies of the different anatomical systems has proven to be a stumbling-block for rapid progress in human avatar modeling research. As implied by the table, modeling the various biological systems of a patient avatar is an arduous task--especially with respect to modeling the physical properties of organs. An increase in Medical VE generations will lead not only to increased geometric fidelity requirements, but also to more complex mathematical models defining the various system responses under varied stimuli. Mathematically defining the characteristics of human biological systems is a particularly challenging task because of the complex interrelationships between physiological systems and the occasionally incomplete scientific knowledge pertaining to how these systems actually function. This is particularly true with respect to modeling Microscopic anatomy and Biochemical systems [SATA96a].

The quality of the patient avatar must also be evaluated in terms of graphical fidelity and adaptability to VE applications. Anatomical data with which to generate high fidelity first-generation patent avatar models is now widely available through many sources. For example, the Visible Human project at the National Library of Medicine has introduced the Visible Human data sets, which provide CT and MRI scans of complete male and female cadavers [LORE95]. In addition, the MAYO Clinic has developed a Virtual Reality Assisted Surgery Program (VRASP) that includes the capability to develop patient-specific avatars from volumetric image data. They are among the first to produce a fully-rendered avatar using the Visible Human Dataset [ROBB96].

The Visible Human and other volumetric data sets are complete enough to generate a realistic representation of a male and female avatar. Unfortunately, the disadvantages with using CT and MRI scans for avatar generation are apparent in the significant amount of segmentation and pre-processing required to generate each model. The resulting size of models generated using these techniques can exceed 150,000 polygons, which makes these models unlikely candidates for use in real-time VE applications without considerable simplification [KERR96].

Because of these limitations and the growing demand for patient avatar geometry, lower fidelity anatomical models are now widely available through third-party sources. These "commercial" models are pre-processed and contain fewer polygons. The lower fidelity models are attractive choices for use in time-critical VE applications. However, they do not contain support for physiological, microscopic, or biochemical modeling. In addition, most do not contain internal details, such as skeletal and organ models.

The demand for high quality patient avatars will not decrease, however. Practitioners eager to develop medical VR simulators are beginning to call for the development of patient-specific avatars that may be used for practicing treatments (primarily scheduled surgery) a priori [ROSE96b]. Thus, the technological demands associated with high-quality patient avatars will likely persist. First- and second-generation patient avatars are now widely available. The Engineering Animation Virtual Human is an excellent example of a second-generation avatar, which incorporates texture maps, NURBS surfaces, skeletal joint interaction, joints, deformation models, and biomechanical models [SELL95].

2.4.1.3 Hardware Limitations Facing Medical VEs

Usability problems caused by limitations in computer hardware are common to almost all medical VE applications. This is primarily due to the special technology required for VE simulations, such as head-mounted displays (HMDs), Liquid Crystal Displays (LCDs), and infrared and magnetic position tracking devices. The following concerns are prevalent in the literature:

- Display devices. Many computer displays lack the resolution required for medical applications. Displays found in HMD devices are typically LCD-based and offer less-than-optimal clarity and resolution [DUMA93; PEUC95; SATA93b; SATA94]. HMD devices are also

typically very heavy and uncomfortable to wear for extended periods of time [PEUC95; POST96b].

- Tracking and input devices. Position tracking equipment and custom input devices are often slow and sluggish [SATA93b; SATA94] In addition, the precision of many tracking devices may be insufficient for some medical VE applications [PEUC95; POST96a].
- Physical actuators. There is a general lack of commercially-available actuators for tactile and force feedback input. These devices are required to correctly simulate contact with patient avatars and medical apparatus. Medical virtual objects must be manipulated with precision, which is difficult without the feedback of real-time interaction [DUMA93; POST96a].
- Graphics rendering performance. Perhaps the most troublesome hardware problem is the inability of current graphics hardware to support the enormous computing demand required by photo-realistic medical simulations [DUMA93]. The state of computer technology has not satisfied the processing requirements of a truly realistic surgical simulator. It is estimated that a rendering throughput in excess of 500,000 polygons per second is required to display a realistic reconstruction of a human abdomen, which is about five times more than typical high-end graphics workstations (circa 1996) can support [SATA93b; SATA94; POST96a].

2.4.2 Emergency Medical VE Requirements

A fully mature emergency medical VE will require a considerable amount of research. The ideal system requirements of the ideal emergency medical VE are defined by Dumay, and are summarized in Table 2-3 [DUMA96].

As shown in the table, the current maturity of anatomic geometry, triage protocols, and texture models is high. All are well understood and currently available. The maturity level of dynamic databases to mange geometry, 3-D visualization tools, and 3-D display devices is adequate and improving. Research is progressing in these areas at a sufficient pace to deliver capabilities in the next three to five years. Research areas that warrant considerable research are the development of models to simulate physical structure and behavior of organs and other objects, and the development of actuator and sensor technologies for force feedback. Thus, many of the well-known medical VE challenges directly impact development of an emergency medical VE.

Requirement	Description	Maturity
Anatomic models	Definitions of outer- and inner-body physical surfaces	High
Physiological models	Descriptions of the structure of organs and muscles	Low
Texture models	Textures and geometry to portray wounds on the body	High
Physical properties	Mathematical model of the physics underlying rigid and soft body deformations and gravity	Low
Triage protocols	Procedures for emergency medical triage and treatment	High
Dynamic databases	Database to manage geometry, levels of detail, textures, and other graphical elements	Moderate
3-D visualization tools	Hardware and software developed for 3-D rendering and visualization	Moderate
3-D display devices	Hardware for 3-D display of models and images	Moderate
Actuators and sensors	Hardware developed to support high resolution, binocular images and haptic force feedback	Low

Table 2-3. Features of an advanced emergency medical VE [DUMA96].

The computational cost of much of the functionality listed in Table 2-3 exceeds the capabilities of current medical VE research. However, the recently explosive interest in medical VR research has spawned many funded projects and motivated many commercial developers to start working on some of these key features. Companies such as High Techsplantations, Inc., Musculographics, Inc., Engineering Animation, Inc., and Immersion Corp. are pioneering hardware and software research to provide detailed physical property software models and new actuators for force feedback and sensory response. This technology is new and very expensive, and prototypes are just beginning to emerge. As discussed in much of the current literature, "the innovators in medical VR will be called upon to refine technical efficiency and increase physical and physiological comfort and capability while keeping an eye to reducing costs for health care. The mandate is complex, but like VR technology itself, the possibilities are exciting and promising [GREE95]."

2.5 *IRIS Performer*

An important aspect of any graphics-intensive simulation system is the management of time-critical rendering tasks. Because the VER is a graphical simulator, management of graphics state is of paramount importance. The VER makes extensive use of Silicon Graphics' IRIS Performer library, a commercial graphics library that provides an efficient framework for developing visual simulations. In IRIS Performer, data structures and functions are organized to streamline the graphics programming requirement by elevating system development to a level of

abstraction above standard IRIX- and Open-GL programming. This section provides a brief description of the Performer library; a more comprehensive discussion is provided in the *IRIS Performer Programmer's Guide* [IRIS95].

The Performer library contains a complete repertoire of C and C++ functions for graphics programming. Internally, the Performer library is composed of several smaller component libraries. The *libpf* library provides a layer of visual simulation functions, database traversal routines, and general graphics state controls. Below *libpf*, and also directly accessible, is the *libpr* library. This library provides low-level graphics functionality and highly optimized rendering functions for groups of geometric primitives. The *libpfdu*, *libpfui*, *libpfdb*, and *libpfutil* libraries use the functions in the *libpf* and *libpr* libraries to provide supplemental database utility support, user interface support, database conversion management, and general utility function support, respectively. The hierarchy of these libraries is depicted in Figure 2-4. Applications developed using Performer are not restricted to exclusive use of the Performer library. If required, IRIX- and/or Open-GL commands may be invoked within such applications as well. Performer is itself based on GL routines and in many cases mimics GL calls that provide similar functions.

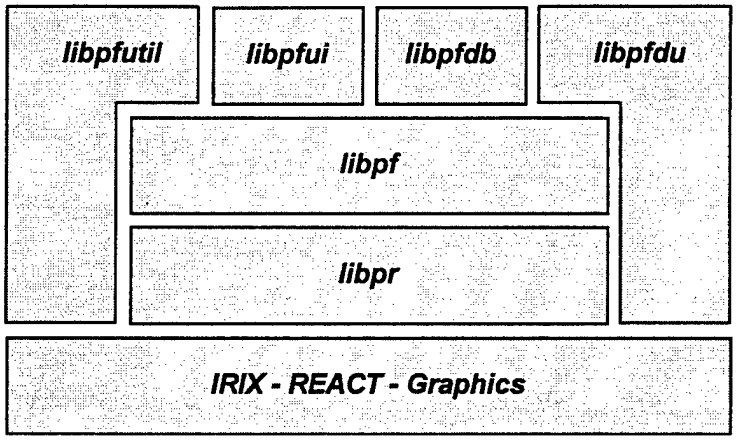


Figure 2-4. IRIS Performer library hierarchy [IRIS95].

If properly configured, Performer-based applications exhibit efficient scene rendering. Performer library routines are optimized to take full advantage of the power of SGI hardware, which may employ multiprocessing to improve performance. These multiprocessing capabilities permit application processing for simulations to be performed in a dedicated "APP" processing

thread. Graphics rendering may be processed in a separately dedicated "DRAW" processing thread. Further, intersection testing and object visibility culling may be performed in a separately dedicated "CULL" processing thread. Performer rendering functions ensure that all available graphics processors are assigned tasks associated with each thread, to the extent that the requisite hardware resources are available. This pipelining, if configured, improves performance and represents one of the key benefits to using the Performer library. If desired, single-processing may also be configured. These processing modes are shown in Figure 2-5.

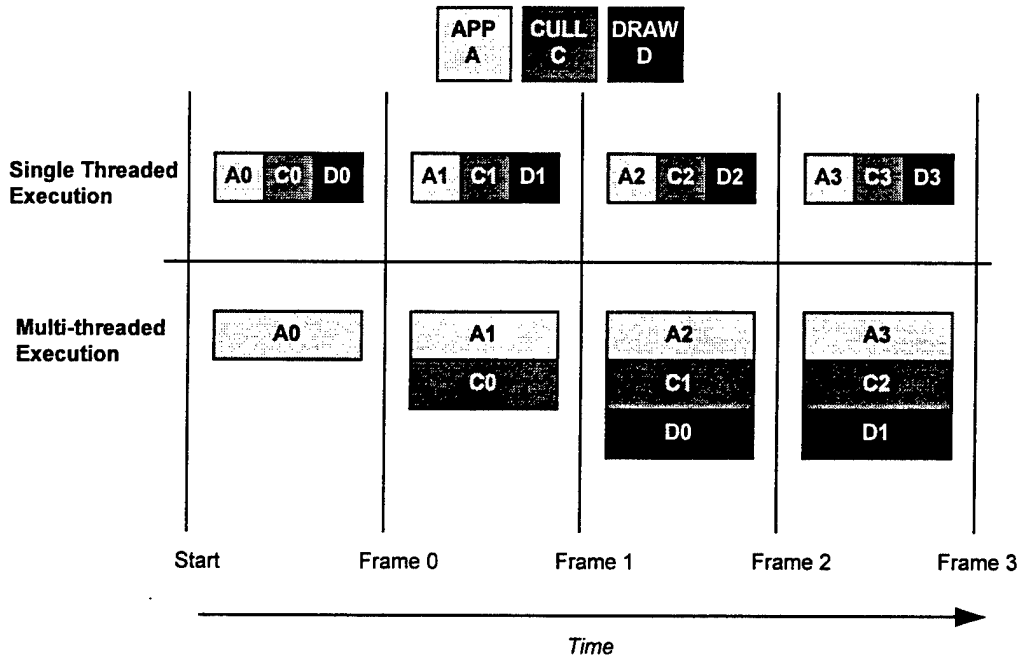


Figure 2-5. IRIS Performer single- and multi-processing [IRIS95].

A particularly useful mechanism that permits the various threads to be programmed is the *pfNodeTravFuncs* library call (*libpf*). This call permits a custom callback function to execute during any of the Performer thread traversals (APP, CULL, or DRAW). A companion mechanism, *pfNodeTravData*, permits accompanying data to be passed into the specified callback function when the callback is invoked. The flexibility of the *pfNodeTravFuncs* mechanism permits callbacks to be executed either before or after the node is processed in the specified traversal thread.

Another beneficial aspect of the Performer library is its strong data-type support for graphics operations. Performer-based simulations depend on the construction and configuration

of an internally-maintained hierarchical database known as a Performer Scene Tree. The Scene Tree is always traversed when the program enters the DRAW thread, and may be optionally traversed during the APP and CULL processing threads. The Scene Tree defines the organization of a VR scene, and may contain several types of nodes as shown in Figure 2-6. Internal nodes define spatial relationships and groupings, while leaf nodes contain geometry that is visible when rendered. Performer also maintains state information in the tree. These nodes, called *geostates*, manage details about materials, textures, transparency, lighting, and other graphics state. The diverse group of specialized nodes is shown in the node inheritance hierarchy.

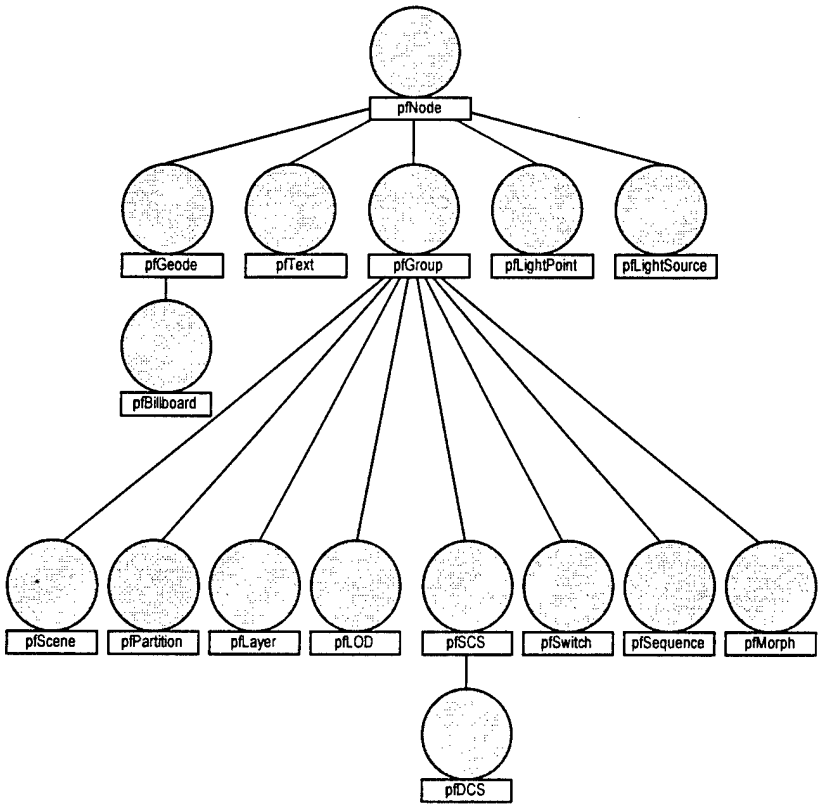


Figure 2-6. Nodes in the IRIS Performer scene hierarchy [IRIS95].

Among the variety of nodes depicted in Figure 2-6, there are several important node types that are fundamental to developing a Performer-based application. The *pfScene* node is a parent node for Performer visual databases. The *pfGroup* node provides a mechanism to group nodes under a common parent in the scene graph, and is primarily used to logically organize the

database. The *pfSCS* node is used to provide a static coordinate system that affects subordinate child nodes. Similarly, *pfDCS* nodes are branch nodes that provide a dynamic coordinate system to change subordinate nodes. Another important node type is the *pfSwitch*, which is a branch node that may be used to select one, all, or none of the child nodes beneath it. The *pfLightSource* node provides a mechanism for introducing lights and lighting effects into Performer simulations. While other node types are useful in other applications, the aforementioned types are of particular interest for developing the VER.

A final strength of the Performer library stems from the ability to utilize a wide variety of geometry database formats. The file support capabilities of Performer are not restricted to a single propriety data format. Routines provided in *libpfdu* can read most popular geometry formats, to include the MultiGen (.flt) format, the Kinetix (.3ds) format, the Wavefont (.obj) format, and the Autodesk (.dxf) format. In addition, several less-popular formats can be imported, such as the Coryphaeus Designer's Workbench (.dwb) format. This "open" approach permits geometry databases in a variety of commercial formats to be easily used.

2.6 Supporting Research

The background information extracted from the literature is not sufficient to create a complete VER design. Information about the emergency medical domain is required. In addition, sources of additional technical information are required. This information was obtained by conducting interviews and discussions with medical practitioners, and on-site visits to emergency medical and technical research facilities.

2.6.1 Interviews and discussions

An important source of information was obtained by interviewing experts. Dr. Godsell-Stytz, a practicing Emergency Room Physician, was available for interview when additional information about the ER domain was required. Further, Dr. Godsell-Stytz directed research to publications that contained supplemental medical information, and was available to critique elements of the VER as they matured. This assistance provided the domain knowledge required to develop the VER from a well-educated standpoint.

Another source of information by interview was a 1-day lecture and meeting with Dr. Richard M. Satava, of the Walter Reed Army Medical Center, and the Defense Advanced Research Projects Agency (DARPA). Dr. Satava is a military physician who oversees DARPA

sponsorship of several medical VR research projects, to include the VER project. His visit to AFIT in May 1996 provided information about how best to proceed with the development of the VER project. He also provided information about other sponsored projects that are investigating related research topics.

2.6.2 Site visits

To develop a complete design, domain-knowledge about the particular layout and function of Level I/II Emergency Rooms is required. Information is also required about the layout of the ER, how the equipment is integrated into the treatment process, and how treatments are administered. This information was obtained from three site-visits to Emergency Rooms in the Dayton area.

The first site visit, to St. Elizabeth's Hospital in April 1996, provided an opportunity to document facility layouts and obtain general information about the inter-communication required within an emergency room. Another site visit, to the Miami Valley Hospital ER in July 1996, provided a glimpse of a more modern, Level I emergency room. Information gleaned from this visit included a more lucid picture of an effective ER layout, and additional information about how emergency medical treatments are administered. A final site visit, to the Wright-Patterson AFB Hospital ER in August 1996, provided information about what a military Level II emergency room contains, and was useful to clarify questions about general ER procedures. Photographs were taken at all three facilities for the purpose of creating the 3-D models required for the VER.

To obtain technical information, a site visit to MITRE's Bedford MA Research Facility in August 1996 was conducted at the recommendation of Dr. Satava. This trip provided insight into how development of a medical facility visualization project is designed and implemented. Not only did this on-site visit provide useful technical ideas, it also generated useful discussion for some of the ideas envisioned for the VER design, and provided a glimpse of other medical VE developments.

2.7 Conclusion

Emergency rooms are staffed with a variety of medical personnel and equipped with a diverse array of medical apparatus. To develop a viable emergency medical simulator, background knowledge about the emergency departments and requisite equipment is essential. The

broad range of cases that must be treated at Level I emergency departments, coupled with the functional complexity of the medical apparatus makes for a VR research topic brimming with research opportunity.

The literature discusses the general notion of developing an emergency medical simulator. However, the literature does not contain detailed research on the subject. In addition, very little work is currently available on existing medical DVE simulation projects. Despite the lack of direct information on emergency medical DVEs, the literature does contain useful design-related information and lessons-learned enumerated by medical VE research teams. These lessons, design criteria, and architectural details are all appropriate for the design and implementation of the Virtual Emergency Room DVE.

The medical VE community is small, very new, and replete with technical challenges. At first glance, it appears that designing and implementing a workable simulation system is an impossible goal. However, as proven with the first Link trainer, "it is not necessary to have ultra-realistic simulators to provide a valuable and meaningful training experience [SATA96a]." Thus, despite technical limitations, the current state of medical VE and patient avatar technology is already capable of supporting a satisfactory and meaningful emergency medical training facility.

3. Requirements

3.1 Introduction

This chapter defines the requirements of the Virtual Emergency Room design. To successfully achieve the research objective, the VER must exhibit certain features and capabilities. These capabilities, grouped as DVE, virtual patient, doctor station, geometry, and support requirements, will lead to the development of a flexible and extensible VER prototype.

3.2 Requirements

The functional requirements for the VER project are categorized into 3 system-level requirements. First, the system must provide an immersive facility in which trainees may practice triage and treatment protocols for administering critical medical care in emergency situations. An immersive training VE permits trainees to see and interact with patients and equipment, thus improving the quality of the intended training. Second, the system must provide a capability to configure and monitor an independent virtual patient process. The virtual patient process must be configurable so that patient information and vital signs are defined prior to each simulation, and are visible throughout the simulation. The capability to monitor the physiological status of the virtual patient is useful in evaluating the effectiveness of simulated treatments. Third, the system must incorporate distributed virtual environment (DVE) technology to provide connectivity between the doctor training VE and the virtual patient process. The addition of a networking capability is fundamental to the notion of using a DVE, and extends system functionality by permitting independent control of the doctor station and the virtual patient process.

After identifying these general system requirements, specific system capabilities and associated design priorities are defined. These requirements incorporate information obtained from discussions with emergency medical physicians and site visits to emergency room facilities. Table 3-1 summarizes the final list of detailed requirements discussed in this section.

3.2.1 Distributed Virtual Environment Requirements

The use of distributed virtual environment (DVE) technology permits VE simulations to support multiple entities. The VER shall exploit the advantages of DVE technology by support-

ing two types of participants: a doctor station that permits trainees to practice emergency medical protocols, and a virtual patient process that receives treatments administered by the trainee.

ID	Detailed Requirement
	DVE Configuration
1.1	Employs suitable DVE architecture
1.2	Independent virtual patient process on network
1.3	Independent doctor station process on network
1.4	Protocol to record and communicate medical events based on DIS architecture
1.5	Strategy to manage communication requirements
	Virtual Patient Process
2.1	Interface to patient physiological model
2.2	Accept treatments from doctor station
2.3	Display current vital signs of virtual patient
2.4	Display simulation performance information
2.5	Dynamically load scenario scripts from external file
2.6	Interactively control simulation parameters
2.7	Interactively select patient avatar for simulation
	Doctor Station
3.1	Trainee immersed in ER setting that permits modifiable view and movement
3.2	Capability to identify and select scene elements
3.3	Capability to move and interact with scene elements during simulations
3.4	Implement functional capabilities of ER apparatus
3.5	Real-time, non-invasive treatments administered by trainee
3.6	Real-time updates of patient monitoring apparatus
3.7	Trainee optionally confined to room and floor using collision detection
	3-D Geometry
4.1	Visually realistic representation of Level I/II ER
4.2	First-generation patient avatar
4.3	Geometry models for ER facility and apparatus
4.4	Positioning and size of all objects is accurate and to scale
	Support
5.1	Accept geometry models in (.dwb) format
5.2	Process approximately 30,000 polygons at a minimum of 10 frames per second with a target continuous rate of 20 frames per second
5.3	Supports input from magnetic position tracker, workstation mouse, and keyboard.
5.4	Operates on existing AFIT graphics lab resources.

Table 3-1. VER detailed requirements.

The choice of DVE architecture is an important consideration. The architecture requires the capability for storing, managing, and accessing doctor-to-patient simulation details. Additionally, the VER requires a means for communicating medical events between the virtual

patient and the doctor station. The communication must be complete enough to clearly relay patient information (such as vital signs data) to the doctor station. Similarly, treatment information generated by the trainee's actions must be communicated to the virtual patient process.

3.2.2 Virtual Patient Process Requirements

The virtual patient process must provide a simulated physiological state for each simulation. Continuously updating this state in response to treatments rendered by the trainee is the basis for VER simulations. Thus, the status of the virtual patient must be continuously available.

In order for the VER simulator to provide a meaningful training experience, instructors and experienced practitioners require a capability to initiate, observe and potentially alter the course of VER simulations. In response to this need, the virtual patient process must provide an initialization and control mechanism for VER simulations. The user must be able to specify patient physiological parameters, select the patient type, and activate the virtual patient process. In addition, the virtual patient process must provide a capability to load patient settings from an external file. Further, a capability to visualize the impact of treatments on the virtual patient is required. A capability to monitor simulation performance information is needed to verify the integrity of the networked environment. A similar diagnostic monitoring capability is also necessary to evaluate the performance of the underlying graphics hardware.

3.2.3 Doctor Station Requirements

The interface of the medical doctor station must be an immersive interface that places the trainee in a synthetic Level I/II emergency room. The environment must contain apparatus geometry models that provide comparable functionality to their real-world counterparts. That is, the trainee must be able to use the instruments to evaluate patient status and render medical treatment using the same mental skills as would be required in an actual ER setting.

Because of the many technological limitations facing "high-end" Medical VE developments discussed in Chapter 2, the VER will not provide a truly hands-on simulation. The many modeling problems associated with object properties and object interaction make modeling of hands-on treatments computationally too expensive. Thus, to develop a workable simulation with existing technology, the doctor station must provide an immersive environment to teach treatment skills at a *procedural* level. Therefore, interaction with objects must occur at a level of

abstraction higher than that required to operate in an actual emergency room. Surgical simulation is not a requirement for the prototype.

The doctor station is a key element of the VER. As a result, a primary requirement is to provide a well-planned human-computer interface [NILA93; NIEL93]. In order for a medical treatment simulation to be useful, it is important that the trainee not be forced into a user interface that complicates use of the application. Rather, trainees should be able to focus on the simulation, without excess concern for how to use the simulator.

Interaction with objects in the VER must be possible insofar as equipment is usable during simulations. Simplifications to the usability of equipment will only be made as a short-term measure to demonstrate interim research capability. In addition, the immersive interface must not include artificial, or "toy-like" capabilities, because such interfaces provide "future reality rather than a useful tool for current procedures [HON96]." Thus, the doctor station interface must not provide information or capabilities not typically available in an actual ER setting.

3.2.4 Geometry Requirements

The graphical fidelity of the VER is another important issue. The VER shall contain 3-D geometric models of a Level I/II Emergency Room facility and most major apparatus. Because realism is an important concern, the geometry databases used in the VER contain texture-maps and materials, to provide an easily recognizable representation of respective emergency room objects. Further, all models must be appropriately scaled and positioned as they would be seen in an actual emergency room. This requirement emphasizes the need for a complete rendition of an actual ER setting.

Due to the limitations of patient avatar development and the associated complexity of physical models discussed in Chapter 2, the VER will employ a first-generation patient avatar consisting of surface geometry only. A first-generation avatar provides a simple initial capability, and is sufficient for demonstrating the feasibility of the VER design.

3.2.5 Support Requirements

The support capabilities required for the VER must be sufficient to support medical staff interactions, patient physiological changes, inbound and outbound messages, and drawing functions in near-real time. That is, system performance must be acceptable in all phases of a

VER simulation. A target average frame rate for the VER is 10 frames per second, and a performance goal of 20 frames per second is desired to support fluid, real-time movement. Based on the geometry requirements, the peak rendering load of the doctor station is estimated at 30,000 polygons per frame.

3.3 Conclusion

The VER must include a doctor station and a virtual patient process as simulation participants. In addition, the DVE design must address the communication mechanism for the underlying DVE architecture. The doctor station must include an immersive interface that permits trainees to view and interact with scene elements as part of the simulation. The virtual patient process must permit control of patient and simulation parameters. High quality, 3-D geometry and adequate support requirements are additional design requirements. The VER design addresses these requirements in Chapter 4.

4. Design

4.1 Introduction

This chapter discusses the VER design developed to satisfy the requirements enumerated in Chapter 3. The design includes a discussion of the DVE architecture used for the VER, and the principal design components, to include a Medical Staff Station, a Patient Control Station, and specialized communication datagrams called MediGrams.

4.2 System Design

The state of current research described in Chapter 2 and the system requirements presented in Chapter 3 represent a functional baseline for the VER system design. This section provides a brief overview of the system-level design. A detailed presentation of the key design features is presented for each main VER design element in subsequent sections.

4.2.1 Design Methodology

The VER system design is based on the Object-Oriented Design (OOD) methodology. The decision to use OOD is based on its many strengths, which include support for data abstraction and sharing of code through inheritance [RUMB91]. Additionally, the OOD methodology emphasizes encapsulation of object properties, leading to a modular design that is both extensible and maintainable. Finally, the OOD methodology provides a natural way to design a system with many real-world components, since each component may be modeled as an independent object. Thus, using object oriented principles to specify the VER design makes the description of the various components easier to understand.

4.2.2 VER Principal Design Components

The VER design consists of three design components, as identified in the requirements:

1. The VER MediGram design, that specifies the format and content of communications between VER simulation participants.
2. The VER Medical Staff Station (MSS) design, that elaborates on the immersive user interface used to train medical staff members.

3. The VER Patient Control Station (PCS) design, that describes how the control station for the virtual patient process is designed.

The VER prototype incorporates more than just these design components. Other required resources include a high-bandwidth local area network capable of supporting at least 10 Mbits per second, and the Medical Network Manager software that permits communications within a DVE to occur at a level of abstraction higher than at the network daemon level [SHEA96].

The motivation behind the design of the VER prototype is to create a DVE that provides a virtual communication channel between MSS and PCS using MediGram messages. The general VER design configuration is depicted in Figure 4-1.

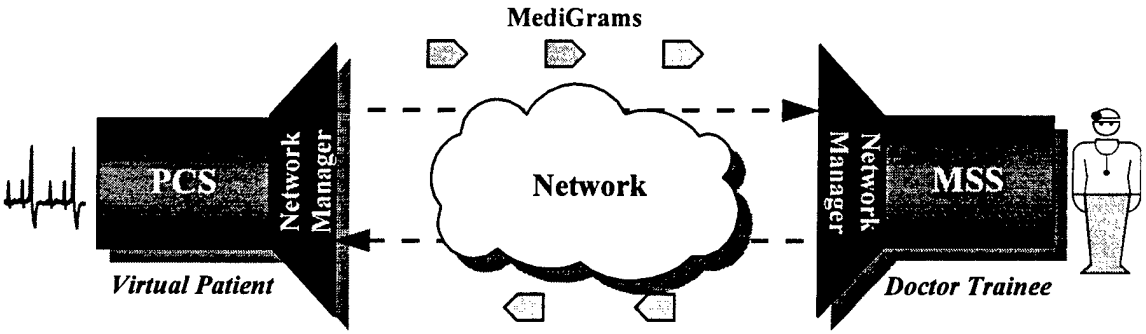


Figure 4-1. VER prototype overview.

4.2.3 DVE Architecture Analysis

An appropriate DVE architecture is required to support the unique simulation requirements of the VER. As discussed in Chapter 2, there are several candidate DVE architectures available. However, two are readily available within the AFIT Graphics Lab that merit consideration: ObjectSim and the Common Object Database (CODB).

4.2.3.1 ObjectSim

ObjectSim is an object framework that supports development and execution of distributed applications. The capabilities of ObjectSim include tight integration with Performer, and several pre-built objects for IO management and graphics rendering. These features, coupled with an installed base of current applications, make ObjectSim a candidate architecture to support the VER design.

There are drawbacks that detract from the usefulness of ObjectSim, however. First, the emphasis of ObjectSim is simulation objects. As a result, the storage and management of simulation data is not emphasized. This is important, because medical applications may be data-driven. Second, the ObjectSim framework is complex. The data structures incorporated into ObjectSim, which are mired in a vast array of Performer library calls, result in a complicated framework. Finally, ObjectSim is designed with aerospace simulations in mind. ObjectSim contains objects that employ aircraft and flight simulation metaphors, which do not apply to other non-aviation simulations. If ObjectSim is modified to remove these features, the object-level integrity of the architecture may be jeopardized.

4.2.3.2 Common Object Database

Another candidate architecture is the Common Object Database (CODB) described in Chapter 2. The advantages of the CODB include direct management of all simulation state in a shared data repository, and a simple but powerful double buffering mechanism that permits simultaneous reads and writes in the database. The emphasis of the CODB is to provide management for a potentially large number of continuously changing simulation parameters. In addition, the CODB provides an unbiased interface that does not presume use of objects developed for a particular purpose.

The drawbacks to using the CODB, instead of a more complete architecture like ObjectSim, are primarily due to the lack of additional support functionality. The CODB architecture is supported by two objects that create and control a shared database. The CODB does not include other support objects such as a renderer and IO managers. However, support functions were developed to demonstrate the capabilities of the CODB and are thus available outside of the CODB architecture.

4.2.3.3 Design Decision

The CODB provides ample freedom to develop data structures and support objects that meet the unique needs of the VER, without forcing medical requirements into aerospace constructs. Thus, the CODB architecture will be used as the basis for the VER design. Support functions developed with the CODB will be adapted for use in the VER design to the largest extent possible.

4.3 VER MediGram Design

Simulation entities in a complete DVE require a capability to communicate events and status to one another. Thus, one design consideration that must be addressed is how the format of the messages should be obtained, and what information they should contain. This section presents the analysis and decisions fundamental to the current MediGram design.

4.3.1 MediGram Format Analysis

The structure and format of the VER MediGrams is a key design decision for the VER system, because the MSS and PCS applications depend on MediGrams to communicate. The concepts upon which DIS communications are based are adopted for MediGrams. However, the format of the messages, which contain medical information, requires additional consideration. There are two alternatives for selecting the format of VER MediGrams: using DIS PDU formats, or developing customized formats.

4.3.1.1 DIS PDUs

One possibility for providing communications in the VER environment is to fully adopt existing DIS PDU formats for different purposes. For example, a DIS Entity-State PDU might be used to pass information about the status of the patient, or the treatment rendered by a trainee. A DIS Detonation PDU might be re-mapped to communicate a patient trauma situation.

The advantage of this alternative is general compatibility with the DIS environment. Although DIS is intended for military simulation, adopting new meanings for existing PDU formats avoids the issue of developing new message formats altogether. This also permits the use of DIS, with the idea that a new special interest group for medical applications might eventually be founded to develop specialized medical PDUs.

However, using DIS PDUs for medical purposes will not permit inter-operability, because PDUs would be interpreted in a non-standard way. In addition, DIS is a protocol that evolves slowly. It could take years before a special interest group for medical DIS applications is organized and agrees on medical PDU formats. Thus, it makes little sense to base VER development on a protocol that does not support the needs of the simulation. It would be difficult, if not impossible, to retrofit the semantics of all possible medical events into the syntax of a military simulation.

4.3.1.2 Customized Datagrams

Another option is to use the strengths of DIS in a DVE communication strategy that directly supports the unique needs of the VER. That is, borrow the architectural concepts from DIS, but create customized datagrams to replace the use of DIS PDUs for communication. In this strategy, the only departure from DIS is the format of the messages.

The use of a special collection of medical datagrams provides ample flexibility for evolving the capabilities of the VER. Special requirements may be supported by tailoring datagram formats. Additionally, changing or adding new medical datagrams is not contingent on approval by special interest groups. Another advantage of this approach is that it does not attempt to force the medical domain on an inherently combat-oriented message repertoire.

However, with this benefit comes two drawbacks. First, the message repertoire will be limited. A completely new definition of required data must be provided, which will take time to evolve. Second, the customized nature of the medical datagrams prevents the VER from communicating with non-VER simulation entities unless the VER additionally adopts support for DIS communication PDUs.

4.3.1.3 Design Decision

Based on this analysis, the VER MediGrams are customized datagrams based on the DIS protocol communication profile. This approach provides the most flexibility for expanding the capabilities of the VER prototype. Intra-VER communication using DIS format PDUs is not a requirement, which provides latitude for developing specialized message formats without undue concern for communication with other DIS entities. The VER MediGrams are focused on medical requirements, but may be expanded to also include useful DIS PDU formats that are not specifically related to military simulations.

4.3.2 High Level Architecture (HLA) Considerations.

The Defense Modeling and Simulation Office (DMSO) is developing a High Level Architecture (HLA) in accordance with objective 1-1 of the DoD Modeling and Simulation Master Plan (DoD 5000.59-P), which was adopted by the DoD in October 1995. According to this master plan, the objective of HLA is "to establish a common high-level simulation architec-

ture to facilitate the inter-operability of all types of models and simulations among themselves and with C4I systems, as well as to facilitate the reuse of M&S components [DMSO96]."

The HLA architecture may be suitable for future VER development because it is not military domain specific. HLA specifies the use of federations to define various types of simulations. Federations contain federate member applications, a Runtime Infrastructure (RTI), and an interface specification between federates and the RTI. The specification of the HLA federation is via a Federation Object Model (FOM), which identifies the essential objects, object attributes, and object interactions that are supported by the federation. The FOM is documented according to an HLA standard Object Model Template (OMT).

According to this arrangement, the customized VER MediGrams may be used in the HLA architecture when it is approved. The definition of MediGrams and specification of the MSS and PCS functionality may be integrated into a Federation Object Model (FOM) for emergency medical simulation. The HLA architecture is evolving closely with the new "DIS++" capabilities, but does not require explicit use of DIS protocols. Thus, the decision to use customized MediGrams does not preclude a near-term migration to HLA.

4.3.3 VER MediGram Design

The contents and format of the customized MediGrams are particularly important. The basic transactions between a trainee consist of three types of information:

1. Time sensitive information about the physiology of the virtual patient, communicated from the virtual patient to the doctor station. This information includes patient vital signs data that are continuously changing over time.
2. Static information about the patient, communicated from the virtual patient to the doctor station. This information includes data such as the age, gender, and blood type of the virtual patient.
3. Time sensitive information about medical treatments, communicated from the doctor station to the virtual patient.

In response to these communication needs, the VER MediGram repertoire includes a ***Patient_Vitals*** MediGram for communicating the physiological status of the virtual patient; a ***Doctor_Treatment*** MediGram for communicating treatments administered to the virtual patient by a trainee; and a ***Patient_Record*** MediGram to communicate information about the virtual

patient and provide startup parameters to initialize simulations. The field definitions of the Patient_Record, Patient_Vitals, and Doctor_Treatment MediGram formats are summarized in Table 4-2, Table 4-3, Table 4-4 respectively. In all applicable cases, enumerations begin with the number 1.

Patient_Record			
type	*size*	*field*	*range*
unsigned short	1	message_id	<RESTRICTED>
short	1	type_id	1 .. 7 [enumerated]
char	52	name	n/a
unsigned short	1	age	0 .. 120
unsigned short	1	gender	1 .. 2 [enumerated]
short	1	blood_type	1 .. 8 [enumerated]

Table 4-2. Patient_Record MediGram format.

Patient_Vitals			
type	*size*	*field*	*range*
unsigned short	1	message_id	<RESTRICTED>
float	1	heart_rate	0.0 .. 300.0
float	1	blood_pressure_systolic	0.0 .. 250.0
float	1	blood_pressure_diastolic	0.0 .. 150.0
float	1	temperature	90.0 .. 110.0
unsigned short	1	pulse_waveform	1 .. 3 [enumerated]
float	1	consciousness	0.0 .. 1.0
float	1	arterial_pressure	n/a
float	1	blood_oxygen	n/a

Table 4-3. Patient_Vitals MediGram format.

Doctor_Treatment			
type	*size*	*field*	*range*
unsigned short	1	message_id	<RESTRICTED>
float	1	defibrillate	0.0 .. 800.0
float	1	heat	70.0 .. 120.0
float	1	anesthesia_quantity	0.0 .. 1.0
unsigned short	1	anesthesia_id	1 [enumerated]
char	1	anesthesia_name	n/a
float	100	IV_quantity	0.0 .. 1.0
unsigned short	1	IV_id	1 .. 3 [enumerated]
char	100	IV_name	n/a

Table 4-4. Doctor_Treatment MediGram format.

4.3.3.1 Patient_Record MediGram

The Patient_Record MediGram is designed to specify information about the virtual patient that does not need to be communicated more than a few times during a simulation. This information, summarized in Table 4-2, includes the following attributes:

1) message_id. The message identification number. Reserved for use by the PCS and MSS MediGram Managers (as described in appropriate sections).
2) type_id. The avatar type of the virtual patient. The values of this field start at 1, and currently range to 7, based on the current patient avatar inventory of the VER. Additional type definitions may be added as additional avatar geometry becomes available.
    1: geometry of the ideal male patient
    2: geometry of the ideal female patient
    3: geometry of the ideal infant avatar
    4: geometry of the obese male avatar
    5: geometry of the muscular male avatar
    6: geometry of the stylized male avatar
    7: geometry of the stylized female avatar
3) name. The name of the virtual patient, represented by an alphabetical string.
4) age. The approximate age of the virtual patient, given to nearest year from 0 to 120.
5) gender. The gender of the virtual patient. Value assignments are:
    1: Male
    2: Female
6) blood_type. The blood type classification of the virtual patient. Value assignments are:
    1: O Positive blood type
    2: O Negative blood type
    3: A Positive blood type
    4: A Negative blood type
    5: B Positive blood type
    6: B Negative blood type
    7: AB Positive blood type
    8: AB Negative blood type

4.3.3.2 Patient_Vitals MediGram

The Patient_Vitals MediGram is designed to specify information about the virtual patient that continuously changes. The primary purpose of this MediGram is to demonstrate a rapidly changing aspect of the virtual patient: the vital signs. The information contained in the Patient_Vitals MediGram, summarized in Table 4-3, includes the following fields:

1) message_id. The message identification number. Reserved for use by the PCS and MSS MediGram Managers (as described in appropriate sections).
2) heart_rate. Specifies the pulse of the virtual patient, on the interval of 0.0 to 300.0 beats per minute.
3) blood_pressure_systolic. Specifies the systolic blood pressure of the virtual patient, on the interval of 0.0 to 250.0 millimeters Mercury.
4) blood_pressure_diastolic. Specifies the diastolic blood pressure of the virtual patient, on the interval of 0.0 to 150.0 millimeters Mercury.
5) temperature. Specifies the temperature of the virtual patient, on the interval of 90.0 to 110.0 degrees Fahrenheit.
6) pulse_waveform. Specifies the shape and pattern of the current heart rhythm wave form exhibited by the virtual patient. The value assignments are defined as follows:
   1: Normal heart sinus rhythm
   2: Traumatic heart sinus rhythm (arrhythmia)
   3: Cardiac failure/inactivity
7) consciousness. The degree of consciousness of the virtual patient, subjectively normalized on the interval 0.0 to 1.0. This value provides an indication of the alertness the virtual patient is, with 0.0 unconscious and 1.0 fully alert.
8) arterial_pressure. Specifies the arterial blood pressure of the virtual patient.
9) blood_oxygen. Specifies the blood oxygen level (SaO_2) for the virtual patient.

4.3.3.3 Doctor_Treatment MediGram

The Doctor_Treatment MediGram is designed to communicate treatment details from the doctor trainee running the MSS to the virtual patient maintained at the PCS. This MediGram demonstrates the capability to render non-invasive treatments from the ER that may affect the physiology of the virtual patient. The Doctor_Treatment MediGram represents a sample of the

treatments possible from within an actual ER. The treatment information, summarized in Table 4-4, includes the following fields:

1) message_id. The message identification number. Reserved for use by the PCS and MSS MediGram Managers (as described in appropriate sections).
2) defibrillate. Specifies a defibrillation treatment, on the interval of 0.0 to 800.0 Joules.
3) heat. Specifies an external heating treatment, on the interval of 70.0 to 120.0 degrees Fahrenheit.
4) anesthesia_quantity. Specifies an anesthesia treatment amount, normalized on the interval 0.0 to 1.0.
5) anesthesia_id. Specifies an anesthesia treatment substance. There are currently no anesthesia treatment substances listed for this enumerated field.
6) anesthesia_name. Specifies the name of the current anesthesia, in alphabetic string format.
7) IV_quantity. Specifies an intravenous infusion treatment quantity, normalized in the interval 0.0 to 1.0.
8) IV_id. Specifies an intravenous infusion treatment substance. The values are enumerated as follows:

    1: Placebo

    2: Sodium Chloride 5%

    3: Sodium Chloride 10%

9) IV_name. Specifies the name of the current intravenous infusion treatment substance, in alphabetic string format.

4.4 VER Medical Staff Station (MSS) Design

4.4.1 Design Overview

The purpose of the VER Medical Staff Station (MSS) is to provide an immersive training experience for ED medical staffs. The MSS is a key element of the VER design. At the simplest level, the MSS accepts inputs from the doctor-trainee, receives MediGram inputs from other simulation entities, and updates the logical and graphical state of the simulation.

4.4.2 MSS Context Diagram

Figure 4-2 shows a context diagram of the MSS component. The trainee inputs, according to the requirements, are from standard workstation mouse and keyboard devices. The MediGram inputs are limited to those generated by the virtual patient (discussed later in this chapter). The Patient_Vitals MediGram is used by the MSS to update the apparatus within the VER environment. The Patient_Record MediGram is used to initialize the simulation and set initial states of VER apparatus.

In the simplest terms, the MSS converts inputs received from MediGrams and the doctor trainee into new MediGrams and scene updates. The Doctor_Treatment MediGram is generated as a result of the trainee's actions in the immersive MSS environment. Beyond communication with other entities, the MSS must continuously update the status of the graphics hardware on the host workstation. Thus, rendering and display outputs to graphics hardware are shown as efferent control flows.

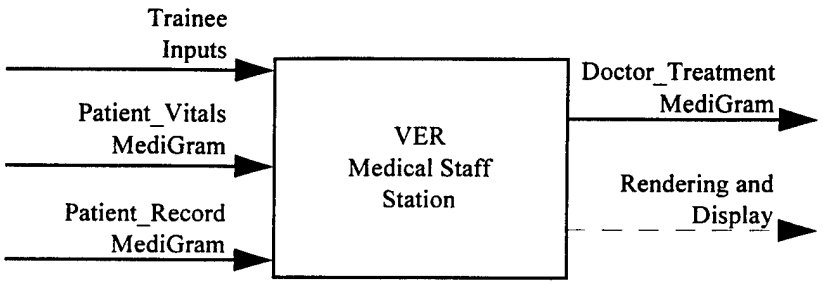


Figure 4-2. VER MSS context diagram.

4.4.3 MSS Block Diagram

A block diagram that shows the primary design components of the MSS is presented in Figure 4-3. The Medical Staff Station uses the AFIT CODB architecture to manage all simulation state. As previously mentioned, the CODB consolidates simulation state information that must be shared with other components of the MSS.

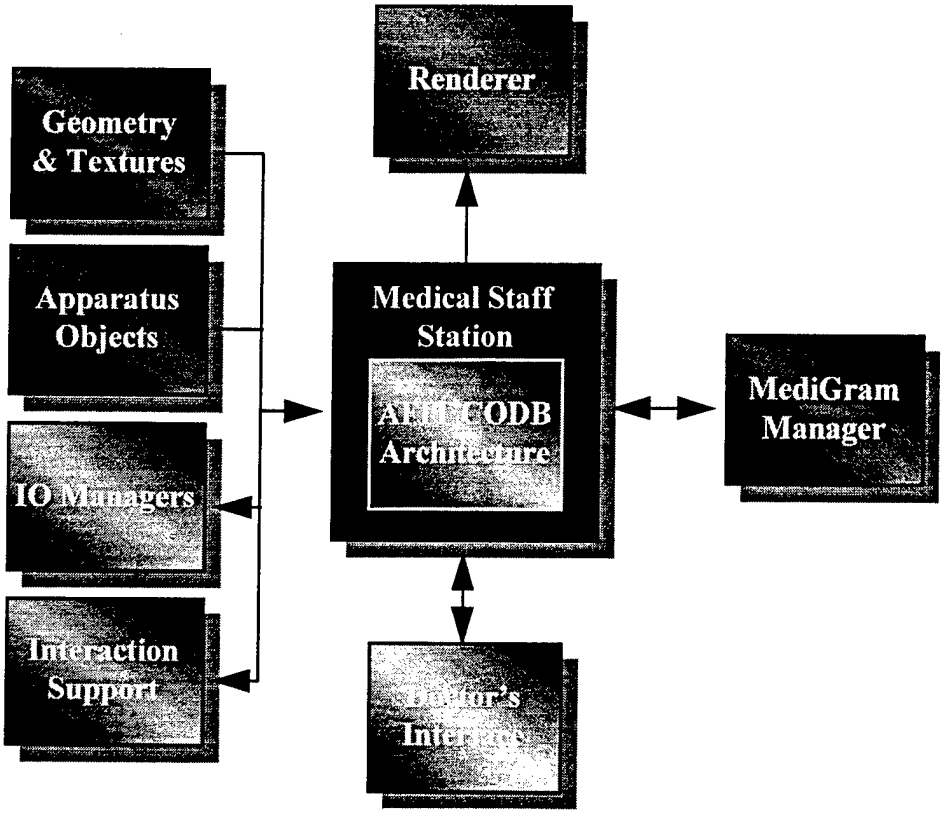


Figure 4-3. VER MSS block diagram.

The MediGram Manager component provides an interface to the Medical Network Manager. The MSS relies on the MediGram Manager to accept inbound Patient_Vitals and Patient_Record MediGrams from the PCS. In addition, outbound Doctor_Treatment MediGrams are queued by the MediGram Manager inside the CODB until they are broadcast by the Medical Network Manager.

The Geometry and Textures component represents the 3-D models and texture maps designed for the synthetic ER scene, to include facility layout and medical apparatus geometry. The Renderer component configures the Performer graphics subsystem, provides an initial workspace with which to develop a Performer-based application, and renders the geometry and other graphical elements of the MSS.

The functional characteristics of each of the 3-D apparatus models is provided by one or more specialized apparatus objects. Apparatus objects provide a means to control the 3-D geometry, automate apparatus control panels, and create Doctor_Treatment MediGrams. The

apparatus objects are also used to access the CODB for the most current Patient_Vitals and Patient_Record MediGrams, so that the apparatus displays in the ER are updated.

Interaction with the immersive ER is made possible by IO Managers that collectively provide the capability to read keyboard, mouse, and magnetic sensor inputs. Inputs received by the IO Managers are interpreted as input events that are stored in the CODB and used to control the simulation. A consumer of input events is the Interaction Support component, which provides the capability to use input events stored in the CODB to select and manipulate objects in the scene. Similarly, the Interaction Support Component uses input events to change the location and orientation of the trainee's view in the ER, and to update simulation parameters.

4.4.4 MSS OOD Object Model

The block diagram of Figure 4-3 may be decomposed into an object design for the MSS architecture. The object model for the VER MSS is presented in Figure 4-4 and Figure 4-5. The object model shows the design of the system as an integration of individual objects. The connections show the object relationships, to include inheritance and aggregation properties. Many of the objects are carried over from the block diagram discussed in Figure 4-3. The portion of the object model shown in Figure 4-4 shows all objects in the design, except for the apparatus objects. Figure 4-5 shows how the apparatus objects are integrated into the design.

The object model diagrams illustrate the extensive use of the Common Object Database. Data structures for each element are supported by the CODB, permitting the state of various objects to be shared as needed. A brief description of the functions performed by each object follows.

4.4.4.1 VER Medical Staff Station Object

As shown in the object model diagram description of Figure 4-4, the MSS process is the primary MSS object. The MSS simulation starts and terminates with the VER MSS object.

4.4.4.2 Medical Network Manager Object

The Medical Network Manager is a VER-specific version of Sheasby's DIS-compliant Network Manager. The purpose of the Medical Network Manager is to provide a black-box interface to commonly used datagram multicast send and receive functions. The Medical

Network Manager uses the CODB as a staging area for managing in-bound and out-bound MediGrams.

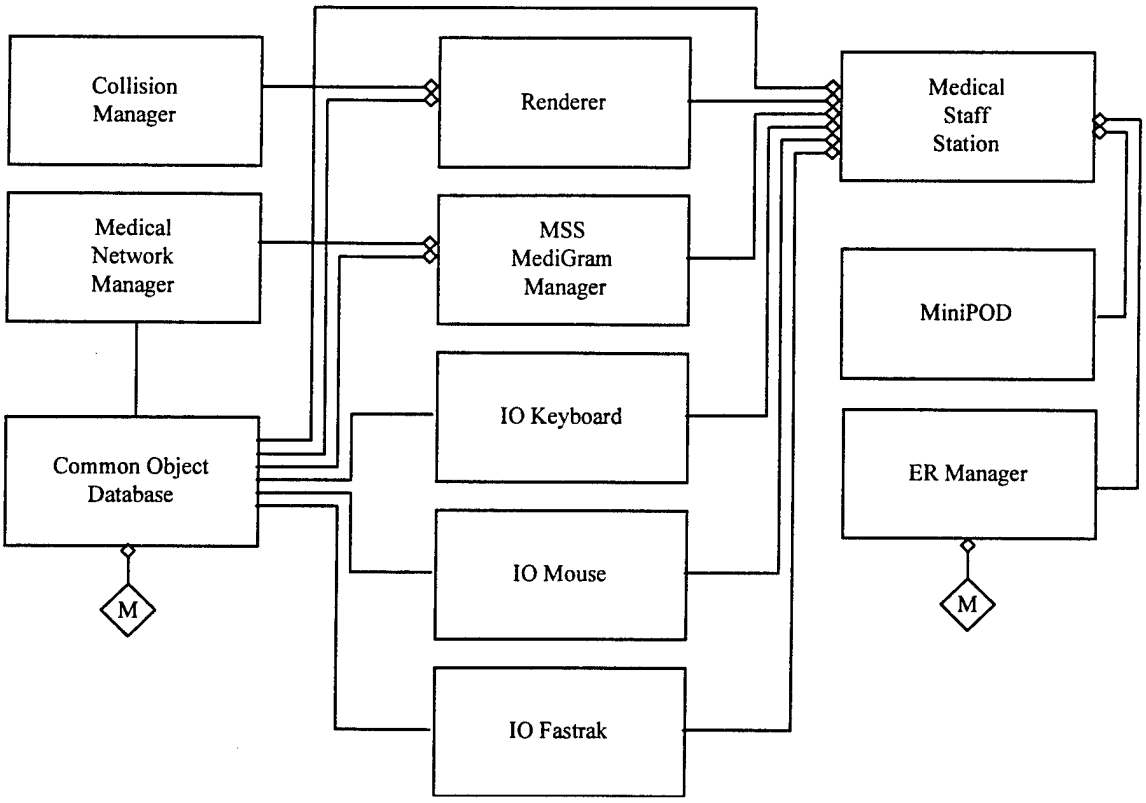


Figure 4-4. VER MSS object model, part 1 of 2.

4.4.4.3 Medical Staff Station MediGram Manager Object

The Medical Staff Station MediGram Manager provides an interface for controlling MSS network communication. This object specifically encapsulates functionality provided by the Medical Network Manager object. Inbound MediGrams from the PCS are received and stored in the CODB by the Medical Staff Station MediGram Manager object. These updates are continuously accessed by apparatus objects that require constant updates. Additionally, the Medical Staff Station MediGram Manager queues Doctor_Treatment MediGrams to broadcast to the PCS.

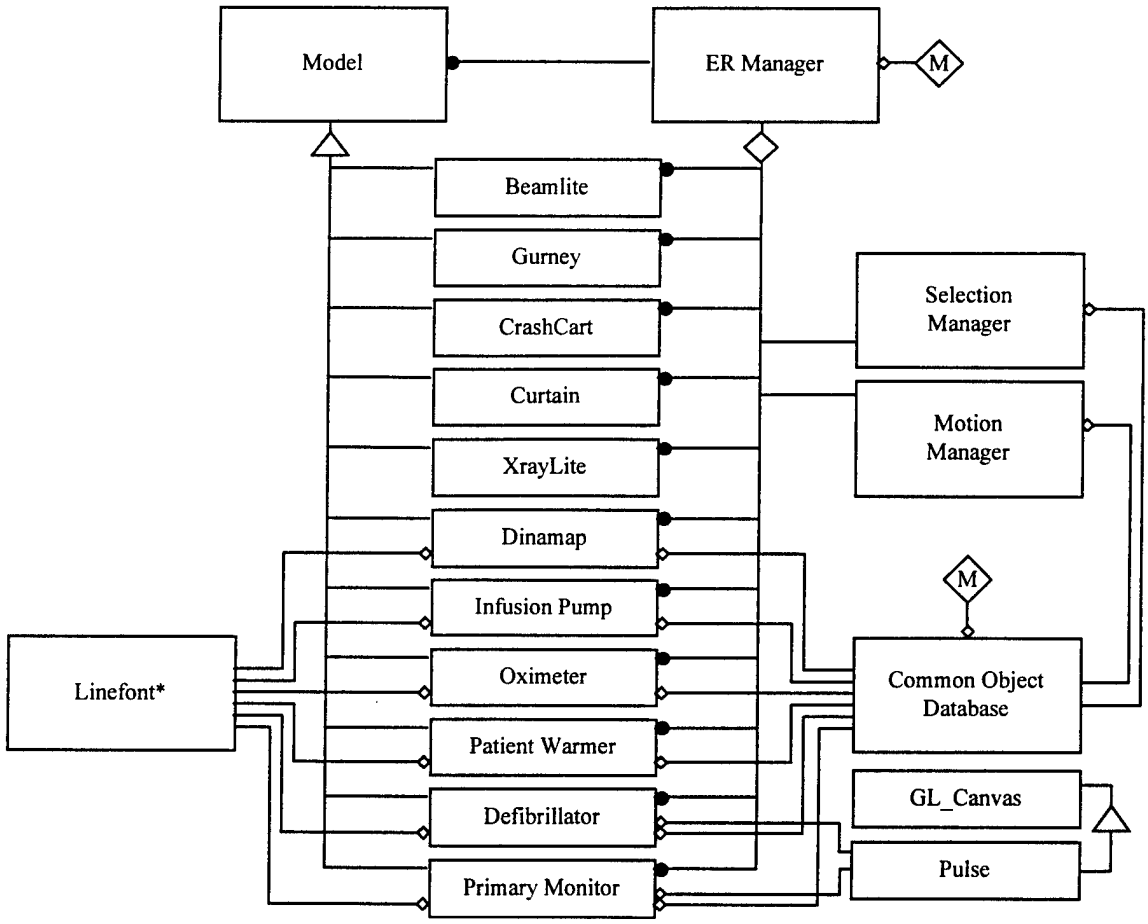


Figure 4-5. VER MSS object model, part 2 of 2.

4.4.4.4 Selection Manager Object

The Selection Manager provides a mechanism for identifying the element in the scene with which to interact. The primary goals of the Selection Manager are to interpret scene element selection commands received from the user, process those commands to identify scene elements, and provide feedback to the user by visually highlighting the designated element.

4.4.4.5 Motion Manager Object

The Motion Manager provides a basic capability to translate and rotate scene elements within the MSS. The Motion Manager is a shared facility used to update the Performer *pfDCS* nodes of the currently selected scene element. Inputs accepted by the Motion Manager object will be mouse-based, because keyboard inputs do not provide information fast enough to permit

smooth motion performance. The Motion Manager includes the capability to accept inputs from a collision detection evaluation mechanism, when such a capability is available.

4.4.4.6 Renderer Object

The Renderer object permits graphical elements in the scene to be drawn on the appropriate display output device. The Renderer configures the Performer graphics pipeline, channels, and windows. In addition, the Renderer manages the state of the user view during the simulation. Thus, all movements within the immersive ER are processed by the Renderer object. A final responsibility is evaluation of user inputs stored in the CODB, to determine what simulation-control actions to perform.

4.4.4.7 Collision Manager Object

The Collision Manager provides a mechanism for integrating a basic collision detection algorithm into the MSS. The collision detection capability provided by the Collision Manager object serves three functions:

1. Ensures trainee's view in the immersive ER is confined to a fixed height above the floor.
2. Provides rudimentary collision detection to prevent trainee's view from moving through objects in the immersive ER.
3. Provides a starting point from which to implement additional collision detection support, such as collision detection between objects in the immersive ER.

4.4.4.8 IO Manager Objects

The IO Managers provide the support necessary to read inputs from various devices and write them to the appropriate CODB structure. In particular, the IO Keyboard object writes keyboard events to the CODB. The IO Mouse object write mouse events to the CODB. The IO Fastrak object writes position data read from the Polhemus Fastrak magnetic sensors to the CODB. All IO Manager objects originally designed to support the AFIT CODB feasibility demonstration in 1996 are adopted for use in the VER.

4.4.4.9 ER Manager Object

The various apparatus objects, as well as the manager objects, must be integrated into a coherent structure. The ER Manager accomplishes this by initializing all apparatus objects,

activating apparatus objects that have been selected by the Selection Manager, and coordinating use of the Motion Manager to move selected objects.

4.4.4.10 Model Object

An important design decision is how to manage external geometry databases in a non-combative, medical simulator. The original option considered was to reuse the model management objects developed for other applications, such as the AFIT Virtual Cockpit. However, after testing these objects, several problems were identified.

First, the model manager object ("fltmodel") only provides the capability to import databases in the MultiGen (.flt) format, which is not a requirement in the VER. In addition, the object contains several unnecessary methods associated with level of detail (LOD) switching and multiple-model instance support. Finally, the model manager object lacks many of the additional support features needed for general model management in a non-combat simulation. Useful features that are not supported include the ability to dynamically change the Performer sub-tree structure for models with articulations, the ability to display information about the status of each model, and other ancillary capabilities (such as geometry highlighting).

As a result of these shortcomings, a new model management object is incorporated into the VER design. This "Model" object is designed based on the general principles of the "fltmodel" object. However, "Model" omits LOD and model-array support methods, and supports the more general model management requirements of the VER. The Model object provides a robust mechanism to load external geometry files and import them into a Performer-readable tree. This object provides the configuration parameters necessary to support all transformers, and provides other ancillary capabilities, such as highlighting.

Besides serving as a base object for apparatus objects, it is also possible to use Model to manage geometry that does not require additional functionality, such as a wall or a shelf geometry database. This capability permits the Model object to be a recurring aggregate member of the ER Manager. A list of the static geometry envisioned for the VER MSS is shown in Table 4-5.

Apparatus and associated objects		Static Geometry using Model object	
Scene Element	**VER Object**	**Scene Element**	**VER Object**
Crash Cart	Crash_Cart	Biowaste "Sharps" Container	Model
Curtain	Curtain	Ceiling/Roof	Model
Dinamap	Dinamap	Doorway	Model
Directional Light	BeamLite	Floor	Model
Gurney	Gurney	Patient Avatar	Model
Patient Warmer	Patient_Warmer	Patient Monitor	Model
Pulse Oximeter	Oximeter	Sink	Model
Rapid Infusion Pump	Infusion_Pump	Storage Cabinet	Model
Xray Viewing Backlight	Xraylite	Storage Shelves	Model
Primary Monitor	Monitor	Utility Panels	Model
		Walls	Model

Table 4-5. VER MSS geometry and respective support objects.

4.4.4.11 Apparatus Objects

The group of objects named after apparatus in the object model diagram of Figure 4-5 are collectively called the apparatus objects. These objects are designed to inherit attributes from the Model base object. The Apparatus objects expand on these basic capabilities by adding capabilities that are needed for the apparatus they represent. The specific objects are also listed in the left side of Table 4-5, and include the following:

1. Beamlite Object. The Beamlite object manages the geometry for the directional overhead lighting fixtures. The Beamlite object includes methods that permit the light to be turned on and off.
2. Gurney Object. The Gurney object manages the geometry for the patient gurney. The Gurney object includes methods that permit the wheels to rotate in the direction of motion when the gurney is moved.
3. Crash_Cart Object. The Crash_Cart object manages the geometry for the crash cart. The Crash_Cart object includes methods that permit the wheels to rotate in the direction of motion when the crash cart is moved. In addition, methods are included to permit each drawer to be opened and closed.
4. Curtain Object. The Curtain object manages the geometry for the ER privacy curtain. The Curtain object includes methods that permit the curtain to be opened and closed.

5. Dinamap Object. The Dinamap object manages the geometry for the patient vital signs monitor. The Dinamap object includes methods that permit buttons on the front panel to be toggled. In addition, the Dinamap object includes methods that continuously display the patient's vital signs, to include heart rate, systolic blood pressure, diastolic blood pressure, and arterial pressure.
6. Infusion_Pump Object. The Infusion_Pump object manages the geometry for the rapid infusion pump. The Infusion_Pump object includes methods that permit buttons on the control panel to be used to control infusion treatments. This object also includes methods that display current instrument settings on the control panel. Finally, the Infusion_Pump object includes methods that permit the wheels to rotate in the direction of motion when the infusion pump is moved.
7. Oximeter Object. The Oximeter object manages the geometry for the pulse oximeter. The Oximeter object includes methods that permit buttons on the front panel to be toggled. In addition, the Oximeter object includes methods that continuously display the patient's heart rate and blood oxygen (SaO_2).
8. Patient_Warmer Object. The Patient_Warmer object manages the geometry for the patient warming system. The Patient_Warmer object includes methods that permit buttons on the control panel to be used to control patient warmth treatments. This object also includes methods that display current instrument settings on the control panel.
9. Xraylite Object. The Xraylite object manages the geometry for the x-ray viewing light panels. The Xraylite object includes methods that permit buttons on the front panel to be toggled. In addition, the Xraylite object includes methods that cause the appropriate light panel to change visual state, based on the respective toggle value of the control button.
10. Defibrillator Object. The Defibrillator object manages the geometry for the integrated defibrillator monitor. The Defibrillator object includes methods that permit buttons on the front panel to be toggled. In addition, the Defibrillator uses information from the control panel to control patient defibrillation treatments. The Defibrillator also provides a dynamic display of the patient's current heart sinus rhythm on an electrocardiogram (ECG) display. The Pulse object is used to drive the ECG display, and may also be used to drive other ECG displays. The Pulse object inherits functionality from the GL_Canvas object, which provides

a general capability for providing a two-dimensional GL graphics display in a 3-D environment.

11. Monitor Object. The Monitor object manages the geometry for the primary patient monitor. The Monitor object includes methods that permit buttons on the front panel to be toggled. The Monitor provides a dynamic display of the patient's current heart sinus rhythm on an electrocardiogram (ECG) display, the patient's current heart rate, systolic and diastolic blood pressures, temperature, and other non-dynamic information such as name, gender, and blood type. As with the Defibrillator, the Pulse object is used to drive the ECG display.

All Apparatus objects are aggregate members of the ER Manager object, which provides updates to each object every frame. In addition, many of the apparatus objects must also access the CODB, in order to store information about treatments administered by the trainee or to update monitors from an updated Patient_Vitals MediGram.

4.4.4.12 LineFont Library

This library provides an IRIS GL-based font mechanism that is shared by all apparatus objects requiring GL text displays. This particularly includes those apparatus objects with integrated control panels. The LineFont library is a C library developed by Wright Laboratory and adapted for use in the VER design.

4.4.5 MSS OOD Dynamic Model

The dynamic model is presented in the State Transition Diagram in Figure 4-6. This model shows the various states that the MSS application assumes during the course of a simulation. The simulation states are depicted as a main simulation loop, which is how Performer requires simulations to be structured.

When the MSS is activated, the processing is performed on one or more processors (and corresponding processing threads), as fast as permissible by the underlying hardware. If multi-threaded execution is enabled, the processing associated with some states may be pipelined across thread boundaries to increase the speed of execution. The processing threads that handle each state in the main processing cycle are annotated adjacent to the state boxes in Figure 4-6.

Initially, the MSS is activated and enters the "Initialize Simulation" state, in which all object instances are created and all simulation state initialized. In addition, a networked simulation requires an initial Patient_Record MediGram be received to synchronize the MSS applica-

tion with the virtual patient (PCS). Thus, the MSS may be in the "Initialize Simulation" state for an indefinite period of time.

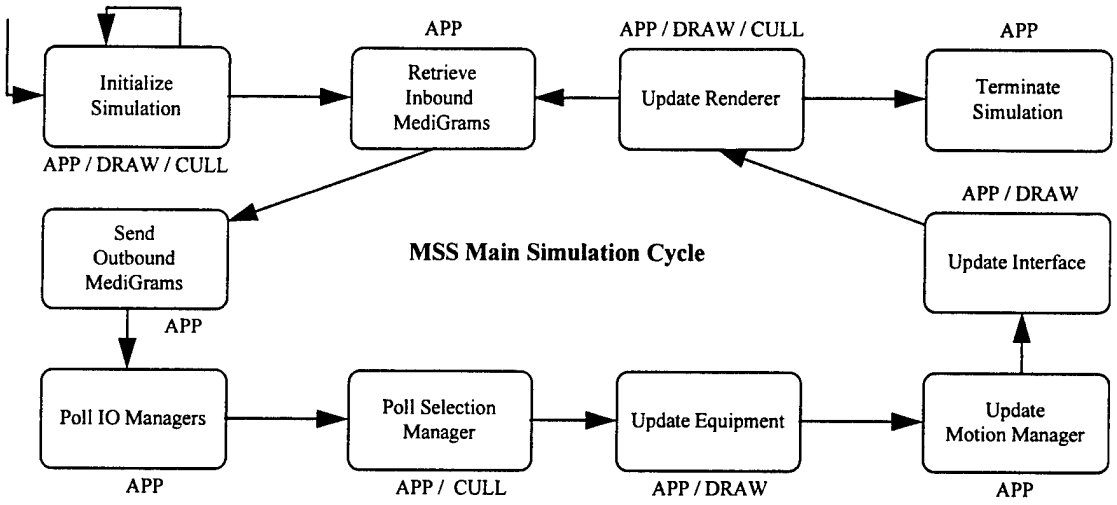


Figure 4-6. VER MSS dynamic model.

The first states to be reached in the iterative portion of the main simulation cycle are the "Retrieve Inbound MediGrams" and "Send Outbound MediGrams" states. These states handle activation of the Medical Staff Station MediGram Manager object for each processing cycle. In the "Retrieve Inbound MediGrams" state, the CODB is checked for new MediGram updates from the PCS. In the "Send Outbound MediGrams" state, the time since the last Doctor_Treatment MediGram multicast send is evaluated. If the time is greater than a specified time threshold, the Doctor_Treatment MediGram queued in the CODB is sent.

After processing MediGram updates, all input devices registered with the MSS are polled in the "Poll IO Managers" state. Events are written to the CODB by the appropriate IO Manager. Interpretation of these events is the focus of the rest of the states. The "Poll Selection Manager" state causes the Selection Manager to be polled to determine if any scene elements are selected. A selected object is identified to the ER Manager in the "Update Equipment" state, which causes the ER Manager to activate and highlight the object.

Control then passes to the "Update Motion Manager" state. When an object is already selected, the Motion Manager reads mouse input events from the CODB and uses them to compute changes in the selected object's position and orientation. Otherwise, the Motion Manager has no effect on scene elements. After processing motion, control passes to the

"Update Interface" state to update the MSS interface. In this state, the CODB is checked for input events that originated in the user interface. If an interface event is detected, that event is processed.

Following this processing, control of the simulation cycle reaches the "Update Renderer" state. The Renderer is invoked to update the scene with the results of the processing accomplished in the current simulation cycle. In the Renderer, the CODB is again used to change the trainee's view position and orientation. In addition, view-level collision detection is performed.

Finally, the "Update Renderer" state includes a check of CODB input events for a terminate simulation event. If a termination event is detected, the simulation exits the main simulation loop and enters the "Terminate Simulation" state to shut down the MSS. If no termination event is received, the Renderer passes control back to the top of the simulation cycle, which then restarts at the "Retrieve Inbound MediGrams" state.

4.4.6 MSS Usability Design

The usability of the MSS environment is an important aspect of the MSS design. Design considerations specifically address three important usability issues:

1. Selection Manager. A mechanism for selecting and identifying scene objects.
2. Motion Manager. A mechanism for moving scene objects.
3. Apparatus Usability. A conceptual approach for interacting with apparatus objects.

4.4.6.1 Selection Manager Design Alternatives

The ability to designate and interact with objects is crucial to the usability of most immersive virtual environments. As a result, it is imperative that the MSS design include an efficient mechanism for selecting objects. The principle objective of the Selection Manager is to permit dynamic selection of objects within the VER, so that selected objects may be interacted with, moved, and otherwise used. The strategies considered to provide this functionality include an interactive menu strategy, a cycling arrow strategy, and a direct channel picking strategy.

4.4.6.1.1 Menu selection strategy

A simple approach for Selection Manager design is to rely on text-based menus to select objects in the scene. In this strategy, a list of selectable objects is constructed and maintained by

the Selection Manager. This list is used to generate a GL or XForms menu that is displayed as an overlay on the scene, which is used to choose an object by name.

The primary advantage of this approach is the direct-selection capability it provides. Users may directly designate objects using the mouse. Another advantage is the simplicity of the interface, which mimics many workstation applications by providing a familiar "pop-up and choose" capability. Another advantage is the simplicity with which this strategy may be implemented. Because this strategy does not require graphics, and because the selection is performed outside of the scene, there is no requirement for additional rendering or graphical pre-processing. Selection is accurate, as long as the name of the desired object is known.

Unfortunately, there are many limitations to the approach. One drawback is the excessively structured means of selection imposed by using a menu. The Selection Manager must be activated and a menu selection must be made to select an object. This strategy leads to an awkward interface, because these steps must be performed every time an object is selected. Empirical tests show this strategy to be distracting, because attention is constantly shifting between the application and the interface.

4.4.6.1.2 Cycling arrow strategy

The cycling arrow strategy permits object selection by cycling a graphical key arrow over the scene elements until the arrow identifies the selected object. The key arrow is positioned using a key-press or mouse-click, and rotates to remain visible regardless of the user's vantage point.

The advantage of this strategy is the discrete number of objects that may be designated. Objects registered with the Selection Manager are included in the selection list that the key arrow cycles through. Objects that are not included are not eligible for selection and are thus ignored by the Selection Manager. An additional advantage is the intuitiveness of this approach, which is graphically more appealing than a simple menu interface. This approach also makes use of entirely graphical elements that are more appropriate for a graphical simulator.

Disadvantages of this approach make it much less desirable. First, the rendering and dynamic orientation of the key arrow requires considerable overhead, because the trainee's view orientation must be used to ensure that the arrow remains visible at all times. This requires excessive computation when other objects obstruct the view of the key arrow. Another problem with this approach is the awkward interface it presents to the user. The Selection Manager must

first be activated, and successfully selecting the intended object may require that the key arrow be advanced several times.

4.4.6.1.3 Direct channel picking strategy

A final strategy for the Selection Manager design involves using the mouse to directly click on the desired object in the visible part of the scene. This requires that a ray be cast from the user's current viewpoint, through the current mouse position on the near clipping plane, and into the current channel viewing frustum. The first intersection of this ray is returned as the selected object.

There are several advantages of this strategy. First, the usability of this strategy is good, because the operation is easily learned and remembered. This strategy also permits direct access to select objects, and the selection process is vantage point independent. Another advantage is the relative simplicity with which this strategy may be implemented. It is a simple matter to cast a ray from the eye through the mouse. Intersection tests involving a single ray are well supported features of the Performer libraries. Finally, the selection of objects is accurate enough to permit very small objects to be selected.

There are also disadvantages to this approach. One problem is the restrictions imposed by the viewing frustum itself. This strategy permits the user to select visible objects only. Because objects that lie outside of the viewing frustum are culled, they are not eligible for selection. This limitation may lead to limited interaction during complicated simulations. Another drawback is the heavy dependency placed on using the mouse. The other alternatives are feasible with other input devices, whereas this strategy exclusively requires the mouse cursor. If other devices must be used for input instead of a mouse, a cursor must be managed and rendered in software. This will lead to somewhat slower results, due to the additional processing requirements added to the APP and DRAW threads.

4.4.6.1.4 Design Decision

Based on the alternatives discussed, the Selection Manager incorporates the direct channel picking strategy. This provides a straightforward and efficient capability for object selection.

4.4.6.2 Motion Manager Design Alternatives

An interface mechanism that permits object movement is another important aspect of usability. Object movement includes rotation about its internal heading and pitch axes, and movement in the x, y, and z world space planes. Two basic mechanisms for providing user-controlled motion are considered for the VER MSS: use of existing Performer functions and customized motion control.

4.4.6.2.1 Performer pfiXformer

One possible mechanism for moving objects is to use the existing *pfiXformer* functions provided in the Performer *libpfui* library. This library contains functions called "transformers" that mimic the motions of driving, flying, and trackball movement. The first two of these transformers are designed to propagate user's view throughout the scene. The trackball transformer, by contrast, is designed to keep the user's view stationary and transform scene elements.

Although the *pfiXformer* provides efficient motion performance, it suffers from a key usability limitation: the *pfiXformer* permits manipulation of objects from a fixed-view only. An observer at the origin gazing along the negative-y axis will find all transformations to be smooth and natural. The mouse events generated to rotate and translate an object are intuitive, because the view is aligned correctly with respect to the *pfiXformer*. However, changes to the user's view diminish the intuitiveness of using the trackball *pfiXformer*. The act of rotating and translating objects is not intuitive, because updates to the top *pfDCS* node of the object are not processed relative to the viewer's position. For example, pushing an object always moves it in the positive-y axis direction, regardless of the viewer's position. This is awkward when the direction of gaze in the scene is in the negative y-axis direction, because a virtual "push" causes objects to come closer, rather than move away.

This limitation is compounded by the fact that the trackball *pfiXformer* does not support collision detection. Further, the *pfiXformers* retain strict control of the *pfDCS* nodes of transformed objects, so it is difficult to override the *pfiXformers* in cases where additional features need to be added. Such features, such as collision detection, are desirable future enhancements

4.4.6.2.2 Customized Motion Control

An alternative approach to transforming objects is to design a customized capability to manage simple object movements. A new motion management mechanism would provide less-

restrictive access to the internal node structure of an object to be moved. By mimicking the node requirements of the *pfiXformer*, a new motion control object provides the ability to introduce collision detection with all ranges of motion provided by the trackball, as soon as good collision detection algorithms are available. In addition, viewer relative features may be added, such as the ability to rotate and translate objects based on the current view position and orientation. With a custom controller, a mouse-generated push may cause objects to intuitively "go yonder," and a mouse-generated pull may cause objects to "come hither." The primary drawback to using a custom controller is the additional complexity added to the design as a result of developing the motion algorithms.

4.4.6.2.3 Design Decision

Based on the analysis of choices, the VER design will include a customized Motion Manager. The Motion Manager will provide simple translation and rotation capabilities by using mouse inputs. The Motion Manager will also permit future collision detection capabilities to be added, which increase the credibility of the motion.

4.4.6.3 User Interaction with ER objects

Another important design consideration is how objects will be controlled, configured, and used during simulations. Four strategies for interacting with ER objects are evaluated: dynamic POD panels, floating control menus, integrated control panels, and XForms pop-up windows.

4.4.6.3.1 Dynamic POD Panels

An initial design possibility is the use of the AFIT POD interface mechanism, which provides a collection of panel and sub-panel objects to develop an integrated interface that "surrounds" the user. The POD interface is used in several AFIT Lab applications, such as the Synthetic Battle Bridge [WELL96], the Satellite Modeler [WILL96], and to a smaller extent, the Virtual Cockpit [ADAM96]. A possible way to adapt the POD to the VER is to develop a front panel that follows the motion and orientation of the trainee's view. On this panel, sub-panels are then added for each object interface requirement. For example, there is a sub-panel for a light and a separate sub-panel for a monitoring device. The sub-panel for the currently selected object is visible, if a sub-panel exists, and that sub-panel provides all of the controls for the object.

The advantages of this approach are straightforward. All of the controls are neatly integrated into a single interface mechanism that is always available to the trainee. In addition, the interface provides a controlled way to interact with the environment by simplifying the number and type of interactions possible—if something cannot be done via the control sub-panel, it cannot be done.

There are several drawbacks to this design approach. First, object interaction typically involves more than merely interacting with control panels. The sub-panel mechanism is, in many cases, an oversimplification of the interaction required in a medical VE. Another similar drawback is the means by which objects are controlled. This interface permits remote control of objects, which introduces an artificial capability not possible in an actual situation. Although this is a procedural simulator, it is important not to resign to over-simplifications of the interface.

Yet another disadvantage of using the POD is the way in which it would be used. Other applications that use the POD rely on it as an observation deck interface for wide-area environments. For example, the Satellite Modeler uses the POD to change views and simulation speeds while navigating through space. Because there are no objects to directly control in space (yet), and the primary goal of the simulation is observation, the POD is a natural fit. However the immersive VER emergency room, unlike the current Satellite Modeler, is an environment that necessitates interaction. A doctor trainee is an active participant, which suggests that a different interface strategy should be used.

A fourth problem with this approach is the mechanism used by the POD to create the interface. POD panels and sub-panels incorporate extensive use of GL, which is not part of the scene, and a model-based panel which *is* part of the scene. This leads to the requirement for a separate mouse driver to interact with the POD panels and sub-panels. By using the mouse driver, additional overhead to resolve mouse contention is required.

A final disadvantage of this approach is the overhead imposed by using the POD architecture. The POD objects are quite large and require a considerable amount of APP thread processing. By avoiding use of the POD architecture, the overall design for the VER MSS may be less complex.

4.4.6.3.2 Floating Menus

Another design possibility involves dynamically creating floating pop-up menus in space adjacent to objects in the scene. That is, the act of selecting a particular object generates an

event to display a pop-up control menu for that object. This control menu is part of the scene, and is directly interacted with to control the selected object. The control menus are based on a simplified POD panel object, called the MINIPOD, which provides a standalone sub-panel.

There are some noteworthy advantages to this approach. First, as with the dynamic POD approach, all of the controls are neatly integrated into a single interface mechanism that is always available to the trainee. In addition, the interface provides a controlled way to interact with the environment by simplifying the number and type of interactions possible. Another advantage is the intuitive nature of this strategy, because control menus are located above or adjacent to the selected object. The menu is visible until the object is no longer selected. Finally, this strategy spatially distributes the user interface according to where the objects are located in the scene. This strategy limits remote control of objects from afar. Users must at least be spatially *near* the objects in order to see and use the interfaces for them, which more closely mimics actual usability.

There are drawbacks to this approach as well. First, this interface is also artificial. User interaction is still limited to use of the MINIPOD panels which are not integrated into the actual apparatus. The MINIPOD panels are derivatives of POD panels and as a result, are included in the scene. Thus, when MINIPOD panels are used to render menus, they appear to hover, presenting an artificial effect. Another disadvantage is the computational overhead required to selectively render each menu panel, and orient it so that it constantly remains visible to the trainee. This overhead is in addition to that required to drive the MINIPOD architecture. Finally, the POD/Performer mouse conflict is a carry-over problem from the POD design alternative.

4.4.6.3.3 Integrated Control Panels

A third design approach involves integrating object control panels into the actual objects for which they belong. Rather on relying on the POD architecture or a new architecture for floating control panels, the interfaces are built into the object models themselves and are always available for use. In this approach, complete object models are constructed with all visual elements integrated into a single geometry database. The integrated control panel is then "extracted" by isolating buttons and other geometry, which are animated to provide a workable interface. GL calls are used to display readouts and other interface details as required.

This approach offers the most flexibility. First, integrated control panels permit the control of objects in a particularly intuitive manner. The best usability approach for the emergency room, as discussed in Chapter 2, is one that most closely emulates real interaction capabilities. This strategy represents a good approximation, because selected objects are accurately controlled only when the trainee is spatially close to the appropriate object.

Another advantage is the integrated nature of the controllable objected. All components are 3-D models, and as a result, all are part of the scene. Unlike a POD-based strategy, there is no problem with simultaneously selecting GL buttons and scene elements. Yet another advantage is the simple overall design that results. There are no requirements to dynamically orient or display panels or sub-panels. All control features, because they are part of the scene, remain visible. The resulting software is also much cleaner, because object objects only incorporate the control functionality needed. There is no inheritance of functions that are not needed.

While this design approach is the most intuitive, there are drawbacks that must also be considered. First, the less restrictive nature of this type of interface does not provide a complete interaction solution. Control panels may be interacted with, but orienting and moving objects is not supported by this strategy. In addition, objects that do not have control panels must also be supported by an alternate interaction strategy.

Another drawback is the more complicated nature of integrating models into the scene. Because the control panels are part of the object, objects must be decomposed into buttons, panels, and other components. This leads to two versions for every model: the complete model as originally designed, and the disassembled version of that model that is actually used in the simulation. For example, the defibrillator was originally modeled as a single entity. With this strategy, all of the buttons must be separated from the frame of the defibrillator so that they may be independently pressed. This decomposition of models complicates configuration management of VER objects and the Performer scene tree.

4.4.6.3.4 XForms Pop-up Windows

A final design possibility involves using the XForms GUI development library to create pop-up control menus. The XForms library is based solely on the Xlib library, and is intended to provide an efficient, easy to use GUI capability with minimal resource requirements [ZHOU96]. Because the interface is X-based, all XForms windows are displayed outside of the Performer scene. Thus, the act of selecting a particular object generates an event to display a pop-up

XForm window in the overlay plane above the scene. The window is then either dismissed by the user explicitly, or is implicitly dismissed when the current object is no longer selected.

There are some noteworthy advantages to this approach. First, as with the MINIPOD approach, controls are integrated into a single interface mechanism. In addition, the interface provides a controlled way to interact with the environment by simplifying the number and type of interactions possible. There are no display conflicts with this approach, because focus on either the simulation or the pop-up control window is controlled with the mouse cursor. Finally, this mechanism provides an intuitive way to interact with objects that would not otherwise have an interface mechanism.

The disadvantage of this approach is the non-immersive interface, which shifts attention between the GUI and the scene. Another drawback is the overhead required to design and integrate XForms into the system, and the requirement to manage system performance as a result of driving the XForms interface. Finally, XForms requires additional work to design and configure the form widgets and callback functions to perform the intended simulation functions within the Performer framework.

4.4.6.3.5 Design Decision

Based on this analysis, the VER incorporates a hybrid approach for object usability. Integrated control panels are used to the largest extent possible, despite the added expense of developing the underlying geometry and support objects. This decision decentralizes the interface to the correct locations in the ER, and minimizes the artificial interface wherever possible. In cases where an artificial interface is absolutely required, such as configuration of objects that do not have controls panels (IV Bags, for instance), an XForms-based interface is used.

4.4.7 MSS Design Summary

To summarize, the MSS design includes the following decisions from the preceding design sections:

- Interface. The interface is both graphical and immersive.
- Selection Manager design. Incorporates the Direct Channel Picking strategy to streamline object selection and interaction.

- Motion Manager design. Incorporates custom control of selected objects, to overcome the limitations of the Performer *pfiXformer*.
- Apparatus usability. Relies on designing apparatus objects with built-in control panels when possible, and relies on XForms pop-up windows for objects that cannot host a control panel.
- CODB environment. All support classes will use the CODB to manage state information.
- Synthetic ER. The MSS hosts the synthetic ER facility for training.

4.5 VER Patient Control Station (PCS) Design

4.5.1 Design Overview

The PCS is the other key element of the VER. The purpose of the PCS is to provide: a process for generating MediGrams that describe the medical status of the virtual patient; and an interface for evaluators and instructors to configure, monitor, and adjust the status of the virtual patient during VER simulations. At the simplest level, the PCS accepts inputs from the user, and receives MediGram inputs from the MSS application. The PCS generates a Patient_Record MediGram to initialize a VER simulation, and regularly generates Patient_Vitals MediGrams to apprise the MSS of the status of the virtual patient.

4.5.2 Graphical User Interface

A preliminary design issue that impacts the PCS is the type of human-computer interface to provide. A preliminary design decision for the PCS is to make the interface graphical, rather than text-based. The rationale for this decision is based on usability concerns. A text-based interface is graphically simple, but may become unwieldy if the number of simulation parameters is large. Similarly, interfaces based on the Curses library or command-line arguments are restricted in their display capabilities. In addition, text-based interfaces lack visual appeal. The wide acceptance of graphical user interfaces has increased general expectation about what a good human-computer interface should include [NIEL93].

4.5.3 PCS Context Diagram

Figure 4-7 shows a context diagram for the PCS component. The Evaluator inputs, like those of the trainee in the MSS, are from standard workstation mouse and keyboard devices.

Scripted input is used to initiate VER simulations from external data files. Inputs are also received in the form of Doctor_Treatment MediGrams generated by the MSS.

The essential processing task of the PCS is to convert user and MediGram inputs into new MediGrams and control display updates. Beyond communication with other entities, the PCS must continuously update the status of the graphics hardware on the host workstation. Thus, rendering and display outputs to control this hardware are shown as an efferent control flow.

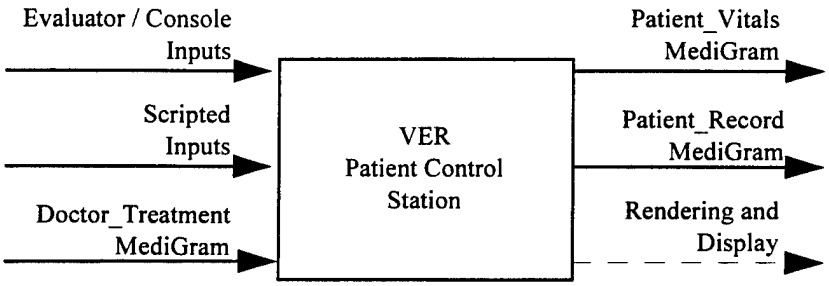


Figure 4-7. VER PCS context diagram.

4.5.4 PCS Block Diagram

A block diagram that shows the components of the PCS is presented in Figure 4-8. The block diagram for the PCS is similar to that of the MSS. The Patient Control Station, like the MSS, is based on the AFIT CODB architecture. All simulation state is maintained within the CODB, which permits sharing between functional PCS components.

Reuse is a key design decision for the PCS. Because the MSS provides many functions that are not specific to the immersive ER requirement, the PCS reuses MSS objects whenever possible. This minimizes the complexity of the design, and permits rapid prototyping. Referring to the figure, the capabilities of the Renderer, IO Managers, Geometry support, and portions of the interface are reused and thus, do not require additional design consideration. In addition, the MediGram Manager is conceptually the same design, but differs in the type of MediGrams that are sent and received. The PCS sends Patient_Vitals and Patient_Record MediGrams, and receives Doctor_Treatment MediGrams.

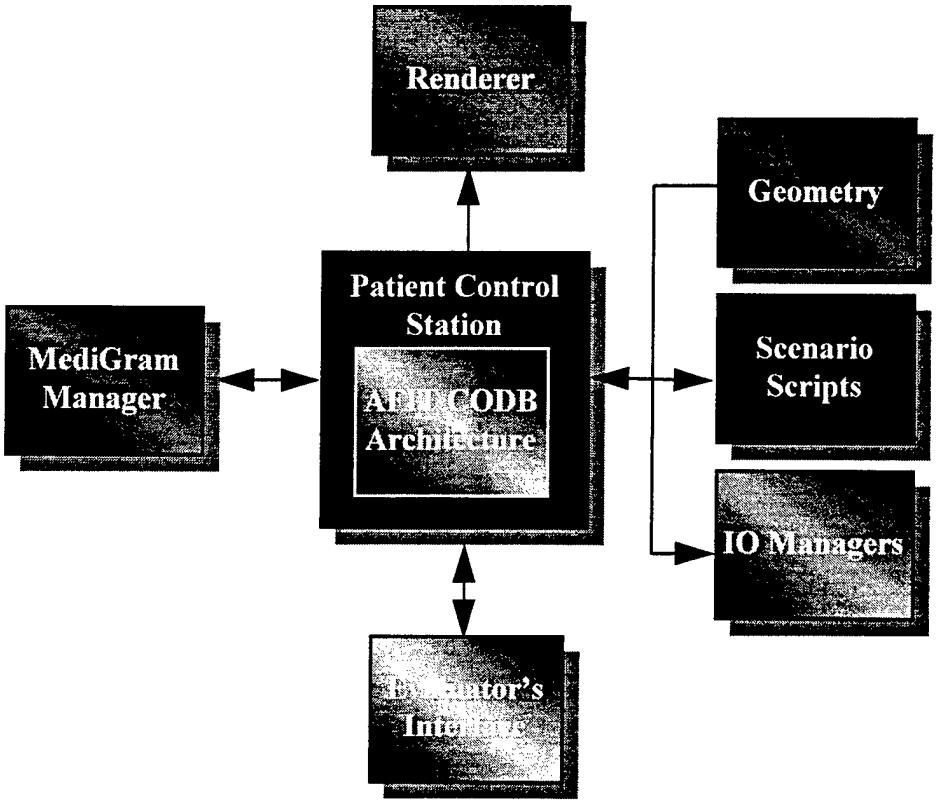


Figure 4-8. VER PCS block diagram.

The PCS also differs from the MSS in the extent to which geometry is integrated into the application. The design requires configuration of the virtual patient avatars. No apparatus geometry or associated support objects are displayed.

The Evaluator's Interface component provides a non-immersive, graphical view of the patient avatar and a control panel with which to adjust patient physiological and simulation control parameters. VER medical scenarios are initiated by specifying the condition of the virtual patient, and then allowing the trainee at the MSS to react to provide required care. Scenario scripts are designed to specify information and initial condition parameters for the virtual patient as a training aid. Information contained in the scripts includes vital signs, text-descriptions of symptoms, background information pertinent to treatment, and other information as required by the simulation. The objective is to provide a flexible means to communicate the initial patient state and the overall treatment objectives for each simulation.

4.5.5 PCS OOD Object Model

The Object Model and Dynamic Model for the PCS application are presented in this section. The object model for the VER PCS is provided in Figure 4-9. Object reuse is made apparent in the object model by annotating reused objects with a star. A brief description of each PCS-specific object follows.

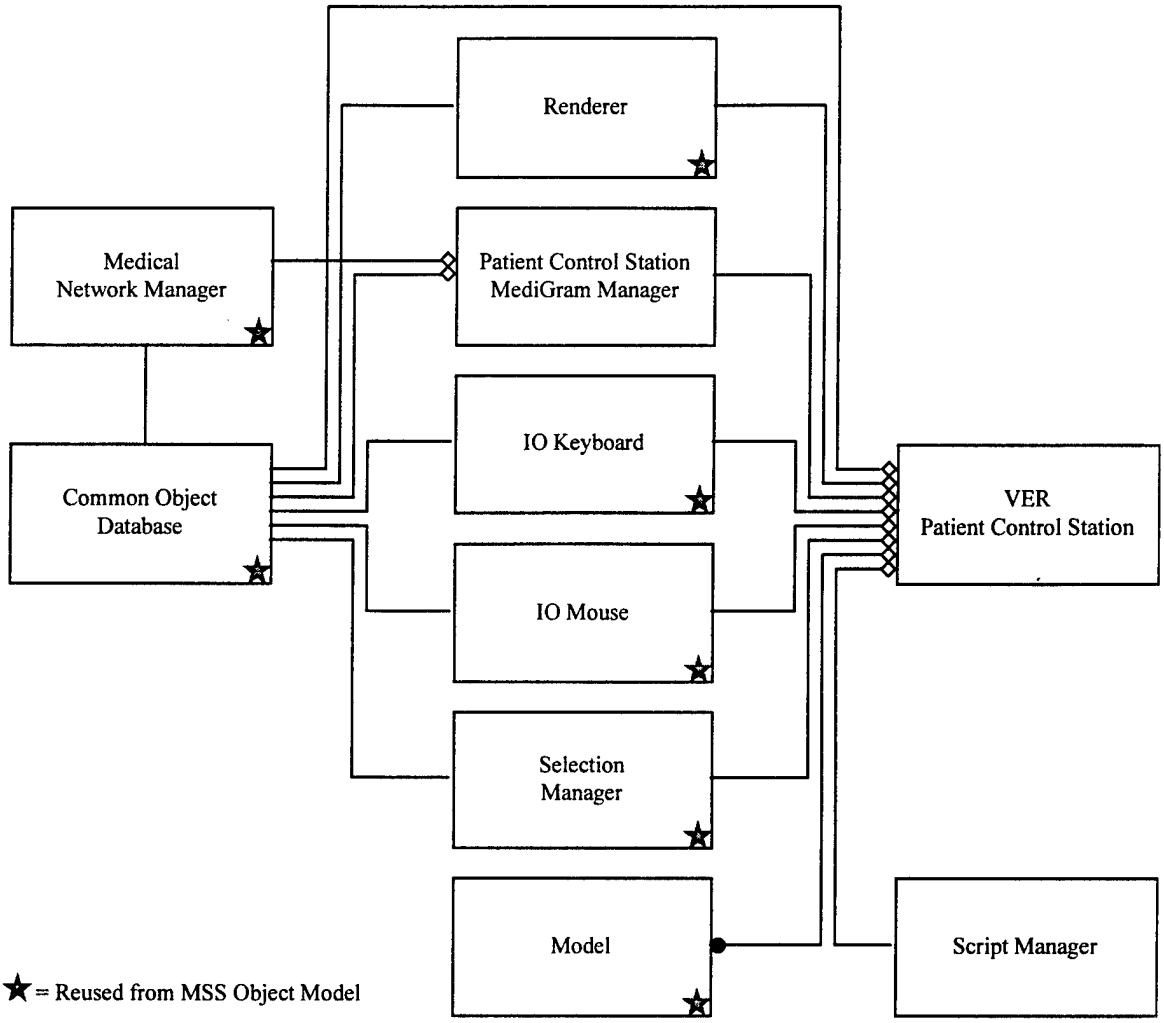


Figure 4-9. VER PCS object model.

4.5.5.1 VER Patient Control Station Object

The VER Patient Control Station Object is the main process for the PCS application. The PCS application starts and terminates within this object.

4.5.5.2 Script Manager Object

The Script Manager object permits the initial physiology of a patient to be loaded from an external script file. In addition, patient information is read from the script file that is used to create a Patient_Record MediGram. Text-based symptom descriptions may also be extracted from the script file in order to provide additional cues for the simulations.

Another important aspect of the script manager is the ability to support a human physiological model, when one becomes available. The object provides an interface into the PCS CODB that permits patient physiology to be modified to would permit the virtual patient to respond to Doctor_Treatment MediGrams.

4.5.5.3 Patient Control Station MediGram Manager Object

The Patient Control Station MediGram Manager object provides network management functionality for the PCS by encapsulating the functions of the Medical Network Manager. Inbound Doctor_Treatment MediGrams from the MSS application are received and stored by the Patient Control Station MediGram Manager. In addition, the Patient Control Station MediGram Manager sends Patient_Record and updated Patient_Vitals MediGrams to the MSS application.

4.5.5.4 Selection Manager

The Selection Manager object is integrated into the PCS application to provide a mechanism for interacting with the patient avatar geometry. Use of the Selection Manager permits the current avatar to be selected and moved for visualization purposes.

4.5.5.5 Evaluator's Interface

The PCS interface is a non-immersive view of the patient avatar and the controls required to initialize the simulation, select a patient avatar, and configure the physiology of the virtual patient.

4.5.6 PCS OOD Dynamic Model

The PCS dynamic model is presented in Figure 4-10. This model shows the various states that the VER PCS object cycles through during the course of a simulation. As with the MSS dynamic model, many of the states in the PCS model are concurrently processed when

executing in a multi-threaded mode. The threads that participate in processing each state are listed by each state in the Figure.

Initially, the PCS is activated and enters the "Initialize Simulation" state, during which object instances are created and initialized. In addition, a networked simulation causes an initial Patient_Record MediGram to be sent to initialize the MSS application. However, unlike with the MSS, the PCS does not enter a hold state to accept an acknowledgment from the MSS.

After initialization, the PCS enters its main simulation loop. Within the main simulation cycle, the "Retrieve Inbound MediGrams" and "Send Outbound MediGrams" states represent calls to the Patient Control Station MediGram Manager object. In the "Retrieve Inbound MediGrams" state, the CODB is checked for new MediGram updates from MSS participants. In the "Send Outbound MediGrams" state, the time since the last Patient_Vitals or Patient_Record MediGram send multicast is used to determine when the next multicast occurs.

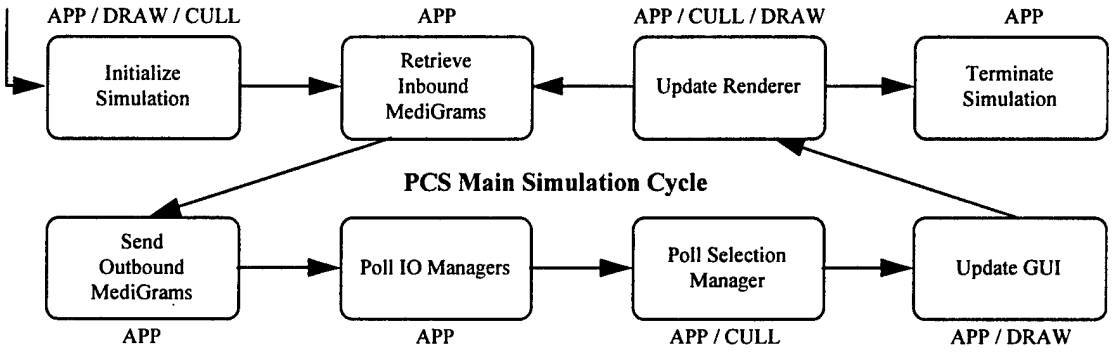


Figure 4-10. VER PCS dynamic model.

After processing MediGram updates, all input devices registered with the PCS are polled in the "Poll IO Managers" state. Input events are written to the CODB by the appropriate IO Manager, and are limited to those received from the IO Mouse and IO Keyboard Managers. The "Poll Selection Manager" state causes the Selection Manager to be polled to determine if any scene elements are selected based on the recently received input events.

The "Update Interface" state is used to check the CODB for input events taken from the user interface. If an interface event is detected, that event is processed. After processing interface events, control of the simulation cycle reaches the "Update Renderer" state. The Renderer is invoked to update the scene with all of the processing accomplished in the current

simulation cycle. The "Update Renderer" state uses the CODB to check for terminate simulation events. The simulation continues processing the main simulation loop until a terminate simulation event breaks the main simulation loop and terminates the PCS.

4.5.7 PCS Usability Design

The PCS usability design is based on analysis of (1) the graphical user interface, for controlling the virtual patient and VER simulation parameters, and (2) simulation control functions and operations that control synchronization between the PCS and MSS applications.

4.5.7.1 Graphical User Interface Design Alternatives

The Patient Control Station interface must permit the PCS application as well as the virtual patient to be controlled and monitored. Three mechanisms for designing the PCS interface are considered: the POD interface, X/MOTIF interface tools, and the Performer *libpfui* interface widgets.

4.5.7.1.1 POD Interface

The POD interface provides a capability to present an interface on one or more panels in a graphical format. The POD employs the rendering requirements of Performer and could provide a basic, immersive PCS interface. This would provide a usability advantage, because features relating to the status of the patient may be drawn or otherwise rendered with relative ease.

A problem with using the POD is that it requires moving within a Performer scene. This adds unnecessary complexity to the design. In addition, there is not a well developed collection of interface tools with which to customize POD panels. Buttons and pull-down lists are possible, but other user interface features (widgets) require additional innovation.

4.5.7.1.2 X Interface Tools

Yet another approach is to use Motif widgets to generate an interface. Interface prototyping tools such as *RapidApp* could be used to develop a flexible control console for the PCS. An advantage to this approach is the widespread familiarity with Motif, consistent and pleasing interface appearance, and the flexibility to modify the interface without considerable innovation.

The problem with Motif is the difficulty of integrating it with Performer. Motif applications may be used to control Performer, but the interface is complex and in many cases difficult to understand. For this reason, Motif is not considered as a viable interface option.

Another possibility for the PCS interface is the XForms library, as described in the MSS design. Although simple and efficient, the XForms interface does not provide enough flexibility to surround Performer scenes in the same window. Rather, XForms are intended to be pop-up interfaces in their own dedicated windows. XForms are therefore not considered, because multiple pop-up windows create a fragmented interface.

4.5.7.1.3 Performer libpfui GUI

A final alternative is to incorporate user interface functions from the Performer *libpfui* library. These library calls permit a user interface to be built around Performer applications. A collection of interface widgets is available with which to integrate the interface.

An advantage to this approach is the capability to create a graphical interface for the PCS that is similar in appearance to many Motif-based interfaces. A single, stationary view is permissible with the *libpfui* widgets. Another advantage of this approach is that the interface is not part of the scene. Thus, there is a distinction between interacting with the scene and interacting with the interface. Moreover, the *libpfui* widgets are developed to work well with Performer applications, so integration difficulties are minimized. A final advantage is the clean appearance of interfaces built with the *libpfui* widgets. The widgets are small, scaleable, and designed to integrate several controls into a relatively small screen area. Thus, it is possible to construct a potentially complex interface without a significant loss of screen area.

The primary disadvantage to this approach is a general lack of flexibility in customizing the attributes of *libpfui* widgets, and the limited number of available widgets with which to construct an interface.

4.5.7.1.4 Design Decision

The PCS design incorporates the *libpfui* GUI interface, due primarily to streamlined integration with Performer. This interface mechanism also permits the PCS to reuse the Renderer object and other Performer based objects from the MSS without modification.

4.5.7.2 Simulation Control Decisions

The control requirements of the PCS extend beyond merely activating a virtual patient process. As the less-complex of the two applications, the PCS is designed to satisfy the added requirements of controlling and synchronizing the overall VER simulation for both applications. In addition, the PCS must provide local performance information to keep the user apprised of underlying simulation details. Thus, the simulation control features will include the following capabilities:

1. Performer performance statistics. Generates an on-screen overlay display of graphics performance for the PCS application.
2. MediGram traffic display. Generates an on-screen overlay display of MediGram traffic that is sent and received, to include current values of the MediGram fields when appropriate.
3. Patient vital signs controls. Permits vital signs to be adjusted over their permissible range. Used to modify vital signs for initialization purposes, as well as during run-time.
4. Patient avatar selection. Permits patient avatar type to be selected before starting the simulation, so that the MSS may pre-load the avatar.
5. Start Simulation control. Permits starting the simulation from the PCS, so that both PCS and MSS applications start in unison.
6. Terminate PCS control. Permits termination of the PCS application.

4.5.8 PCS Design Summary

To summarize, the PCS design includes the following decisions from the preceding design sections:

- Graphical interface. Provides flexible and extensible interface that is visually interesting.
- Script manager. Permits scenario data to be loaded consistently for each simulation.
- GUI interface (libpfui). Provides streamlined GUI interface capability that integrates well with Performer. The interface is non-immersive.
- Reused components from MSS. Provides consistent capabilities across the VER design. This minimizes redundant workload.
- Multi-purpose simulation control. Provides interface capabilities necessary to operate the simulation and control the virtual patient.

4.6 VER Geometry Design Decisions

Quality 3-D geometry is a key design ingredient of the VER MSS and PCS components. To provide consistently designed geometry, design decisions about level of detail switching and geometry database formats are required.

4.6.1 Level Of Detail

In many VE applications, judicious use of Level-of-Detail (LOD) switching permits good performance when the viewing frustum contains many scene elements. The goal of LOD switching is to substitute high fidelity geometry with lower fidelity geometry when the viewing distance exceeds one or more distance thresholds. Thus, the overall polygon count contributed by each object in the simulation is adjusted as a function of the user's viewing distance [VINC95]. Using this technique, graphics performance is preserved because at long distances, otherwise invisible geometric features are not rendered. Geometry files that incorporate LOD support contain several detail levels integrated into a single geometry database file.

LOD switching is particularly well suited to simulations that encompass a wide virtual area, such as space and combat simulations. The viewing distances in such applications can easily encompass hundreds of miles, which makes discernment of geometric details difficult from afar. However, the usefulness of LOD switching is diminished in simulations that are confined to relatively small areas, because details may be discerned at all times [IRIS96]. In fact, LOD switching can adversely impact the rendering performance of close-quarters simulations, because viewing distances vary by considerably smaller amounts. In the context of an emergency room setting, LOD switching may obscure visual details that are needed during simulations. Thus, it is possible that if LOD switching is activated, a view of one or more objects may be incomplete.

Another problem is the ease with which LOD switching is typically noticed. Over long distances, geometry switching is barely noticeable and thus provides a visually acceptable solution. However over short distances, the geometry switching may be stark and potentially distracting [VINC95]. While fading or blending detail levels is possible, the process is still apparent and may incur a performance penalty. This penalty arises from the manipulation and alpha-blending of multiple versions of the geometry required to smooth the switching [VINC95]. Finally, LOD switching complicates the design of the Selection Manager and the simplicity of

the integrated control-panel strategy. The problem lies in knowing which LOD is active at any instant so that it may be correctly processed by other application elements. Based on this analysis, the VER does not incorporate LOD switching geometry.

4.6.2 Geometry Database Formats

The database format of VER geometry is another design issue. Wire frame models are readily available from many sources, some of which are free or very low cost. Performer has a well-rounded capability for importing most common database formats. Unfortunately, the AFIT Graphics Lab does not have a modeling tool capable of editing models in many formats. Coryphaeus' Designer's Workbench supports just 5 formats, including the native (.dwb) format. Thus, a limitation is imposed on the ability to edit non-native geometry obtained from other sources in other formats.

Another concern is the functional limitation of some Performer database loaders. The *pfLoad_dwb* loader is equipped to translate advanced modeling details such as instancing and transparency. However, not all loaders are capable of translating these features. Use of other model formats may not provide expected results within the Performer environment.

Because of these issues, the VER geometry is designed according to the following strategy:

1. Design geometry using Coryphaeus' DWB, and save as (.dwb) format to maintain a consistent level of geometric fidelity, and to ensure Performer compatibility.
2. If the above is not possible or too work intensive, then import external geometry into Coryphaeus' DWB, edit and save as (.dwb) format to ensure compatibility.

This strategy ensures that all geometry may be modified, and dependency on database interpretation by Performer loaders is limited exclusively to the (.dwb) file format loader.

4.7 Conclusion

The primary objective of this chapter is to present the design of the VER prototype. The VER design is influenced by research of not only the current literature, but also medical facilities and other VE applications. In addition, a general philosophy of technical reuse is applied whenever possible.

The designs of the MSS and PCS applications address the functional requirements established in the beginning of this chapter. These applications are designed to operate within a

DVE framework that is directly supported by the Medical Network Manager, and the Common Object Database architecture. The design of the MediGrams, which facilitate communication between VER applications, is based on the strengths of the DIS protocol and compliance with HLA standards. In addition, the applications are designed to use the Performer libraries.

The structure of the design for the VER is not complex, once it has been decomposed into distinct design elements. The manager-driven approach to performing most tasks is easy to work with, and lends itself well to object-oriented constructs. Further, the design is succinct enough to provide flexibility in implementation. A discussion of how this design is implemented, and what implementation-level tradeoffs were made, is presented in Chapter 5.

5. Implementation

5.1 Introduction

This chapter describes the implementation of the VER prototype, based on the design outlined in Chapter 4. An overview of the implementation is first described, followed by a description of classes that are shared in the MSS and PCS implementations. Next, the Medical Staff Station (MSS) implementation is presented. Afterwards, the implementation details of the Patient Control Station (PCS) are presented, followed by a discussion of the processes used to implement the 3-D geometry for the prototype.

5.2 Implementation Overview

The VER implementation, as with the design, describes the MSS and PCS applications. However, classes that provide functionality common to both applications are consolidated into a third element group called the VER "shared classes." Figure 5-1 shows this grouping.

The shared classes include the implementation of general performer rendering capabilities, the CODB classes (the implementation of which will not be discussed, see Stytz for additional information [STYT97]), the IO Manager classes, and the Model geometry management class. Implementation of this functionality is discussed in Section 5.3.

MSS Implementation details are discussed in Section 5.4. The MSS application is controlled by the VER_MSS executive. As depicted in Figure 5-1, the VER_MSS function directly owns and controls the MSS_Renderer class, instances of the IO_Managers from the shared class group, and the MSS_Medigram_Manager class. Each of these classes perform a service that supports the simulation in general, without directly affecting the appearance or management of the synthetic ER facility. Therefore, these classes are grouped as "MSS Support" classes. The ER_Manager, also controlled by the VER_MSS, manages the ER scene and the interaction capabilities permitted within the scene. The ER_Manager directs all processing within the synthetic ER environment. This processing includes initializing and updating apparatus, maintaining the ER Performer scene tree, and processing manipulations of scene elements. The classes that support ER interaction are described under the heading of "Interaction Control."

PCS Implementation details are discussed in Section 5.5. The PCS application is controlled by the VER_PCS main. The VER_PCS main directly owns and controls the

PCS_Medigram_Manager class and the Script_Manager class. In addition, the VER_PCS main uses an instance of the Selection_Manager class to assist with controlling the GUI interface. All other functionality is reused from the shared class group.

Geometry is another important ingredient, common to MSS and PCS implementation. A discussion of the processes taken to implement apparatus and patient avatar geometry is provided in Section 5.6.

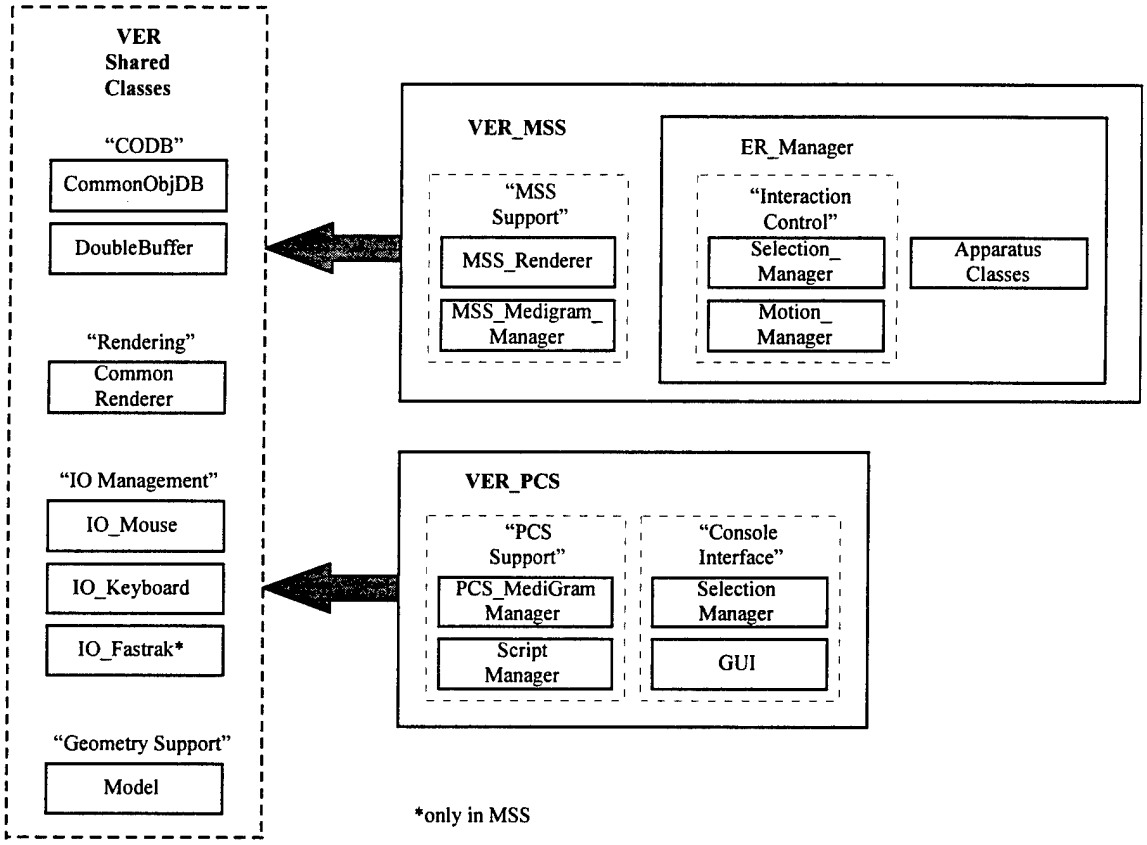


Figure 5-1. MSS implementation overview.

5.3 *VER Shared Classes*

The VER shared classes are implemented to facilitate reuse to the largest extent possible. Because both applications are implemented using the Performer libraries, common Performer-based functionality is implemented as a separate group of classes shared by both applications.

5.3.1 Common_Renderer

The Common_Renderer implements Performer configuration requirements common to all Performer-based simulations. These requirements include configuring the *pfChannel*, *pfPipeWindow*, and all scene traversal functions for the CULL and DRAW threads. In addition, this class configures general environmental settings such as earth-sky attributes, clipping plane parameters, and Performer rendering modes. Thus, this class is predominantly composed of Performer library calls.

5.3.2 IO Managers

The IO_Keyboard class is implemented to store keyboard and window events in the CODB. The IO_Keyboard class relies on the *pfuGetEvents* in *libpfutil* to perform these tasks. The IO_Mouse class relies on the *pfuGetMouse* to obtain all mouse update events. This information is also stored in the CODB for use by other objects. In addition to basic mouse event processing, the IO_Mouse class also contains a method that generates a software cursor using IRIS GL. The IO_Fastrak class is used to configure and manage position updates received from the Polhemus Fastrak sensors. The class contains the source for a library of routines that manage access to the Fastrak server process. The Fastrak server process accesses the serial port and retrieves position data.

The IO_Keyboard and IO_Mouse Classes rely on X device input processing. According to the Performer Programmer's Reference, X device input is the best configuration option because "GL does not contain efficient device input routines; collecting GL device input in the draw process can reduce rendering throughput; and collecting X device input in an asynchronous process can improve real-time characteristics [IRIS96]." Tests of input strategies confirm these statements.

5.3.3 Model class

Management of geometry databases within the Performer environment is crucial to the success of the project. All of the visual scene elements in the VER are external geometry databases—files containing 3-D modeling information not created specifically by Performer functions. The Model class is implemented to provide support functions common to all external geometry databases. The primary capabilities of this class include the following:

- *Standardized Performer sub-tree for each Model object.* The Model object imports geometry databases using one or more *libpfdb* file readers. The basic structure of each model instance includes *pfSCS*, *pfDCS*, and *pfGeoSet* nodes. This configuration supports static geometry requirements, because objects may be positioned and optionally re-oriented. For objects that move within the scene, an alternate structure configuration is also supported. In this configuration, an additional *pfDCS* parent node is attached as the root node of the Performer sub-tree. This permits objects to be moved and rotated by motion support functions, such as the *pfiXformers* in *libpfutil* and the VER Motion_Manager object. This flexible Performer tree structure is shown in Figure 5-2.

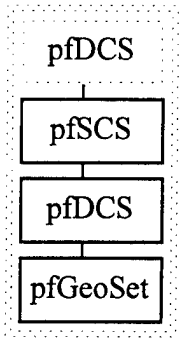


Figure 5-2. Performer tree structure created by the Model class.

- *Node Initialization.* The Model object initializes *pfNodes* in the Performer tree structure with simulation-specific information. In particular, the properties of each object are defined in a static *pfSCS* matrix, which contains scaling information and initial translation and rotation values as offsets from the world-space origin. In addition, the node name of each object is assigned to the appropriate *pfDCS* parent node, and the lighting characteristics of the *pfGeoSet* nodes are initialized.
- *Performer pfDCS Matrix Updates.* The Model object permits rotation and translation of geometry using world-space coordinates. Movement of objects is supported by methods that perform absolute or position-relative translations and rotations. These methods operate on the *pfDCS* nodes in the Performer sub-tree hierarchy. The functionality supports Model configurations with and without the optional parent *pfDCS* node.
- *Model State Information.* Information about the status of any database object is retained by the Model object and may be displayed. Information about the status of each sub-tree is

available through methods in the Model object that print the state of all matrices and file-name details. This information is particularly useful when initializing the simulation, as well as for troubleshooting enhancements to geometry or motion-related capabilities.

- *Augmented Graphics*. The capability to highlight the geometry and associated bounding volumes is implemented using the *pfHighlight* function. The Model class manages the state information necessary to configure and selectively activate the *pfHighlight* mechanism.
- *Inter-operability*. The Model class includes support attributes and functions that support inter-operability with other classes. The most apparent case of this is the built-in support for the Selection_Manager (described in later section).

The class methods that provide these capabilities are shown in Figure 5-3. The Read-Model and Model constructor build the Performer sub-tree from imported geometry database elements. The SetRot, SetTrn, GetRot, and GetTrn methods define and evaluate the Model *pfDCS* nodes for rotation and translation, respectively. The SetRotDelta and SetTrnDelta permit relative *pfDCS* node updates using incremental adjustments.

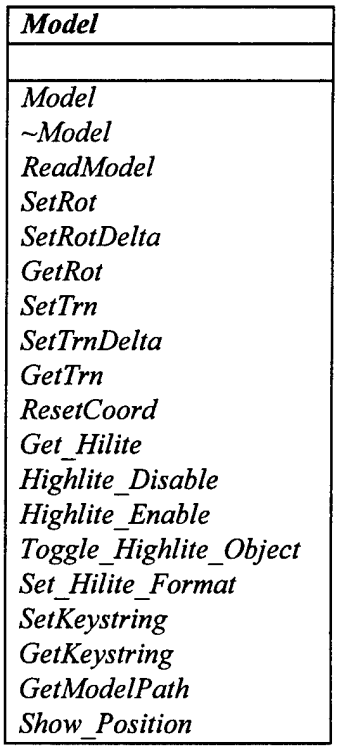


Figure 5-3. Model class methods.

Also provided are methods to get and set the keystring used by the Selection_Manager to identify scene elements. A collection of geometry highlighting support methods are implemented to define, activate, and control the *pfHighlight* configuration. Finally, the Show_Position method displays position and other information about the Model object.

5.4 MSS Implementation

5.4.1 VER MSS Main

As the executive process of the MSS application, the VER_MSS main initializes the application, controls the main simulation loop, and generates a minimal collection of interface displays.

5.4.1.1 MSS Initialization

The VER_MSS Main function provides primary control of the simulation and all Performer functionality. An important aspect of the VER_MSS main is the initialization of Performer and respective shared memory structures. The VER_MSS main defines and creates internal data structures and shared memory structures to manage data across multiple processing threads. The VER_MSS also initializes Performer and all local and shared memory variables.

After Performer is initialized, the VER_MSS creates and initializes all support class instances. Instances are created for the VER_Renderer, IO_Mouse, IO_Keyboard, ER_Manager, and MSS_Medigram_Manager. These classes perform the rendering, IO management, local entity management, and communication functions for the VER MSS application. As part of initializing the MSS_Renderer, the VER_MSS defines callbacks for CULL and DRAW thread processing.

Also performed by the VER_MSS is synchronization with the PCS. The VER_MSS decodes command-line parameters to control the initial mode of execution of the MSS application. The MSS may operate on or off the network. The on-network mode causes the MSS to synchronize with the PCS during startup, which sets the MSS simulation clock and directs the ER_Manager to load the avatar specified by the first Patient_Record MediGram received. Off-network execution causes the MSS application to operate without accepting MediGrams from the PCS application, and does not load a patient avatar.

5.4.1.2 Simulation Control

MSS Processing is controlled from a main simulation loop. The simulation loop in the VER MSS application delegates most of the immersive simulation processing to the ER_Manager, the MSS_Medigram_Manager, and the VER_Renderer classes. However, the main simulation loop of the VER MSS still controls the overall application by accomplishing the tasks shown in Figure 5-4.

```
while (!done)
        Process Panic Requests, to reset view and position
        If on network:
                MSS_Medigram_Manager retrieves PR and/or PV MediGrams
                If time interval since last DT MediGram multicast exceeds threshold:
                        MSS_Medigram_Manager sends current DT MediGram
        Update ER_Manager
        Initiate Cull and Draw traversals
        Poll Mouse
        Poll Keyboard
        Update View in VER_Renderer
        Update GUI bar
end loop
```

Figure 5-4. MSS main simulation loop.

Within the main loop, the VER_MSS continually checks for and processes panic requests to reset the current view. Immediately thereafter, the VER_MSS calls the MSS_Medigram_Manager to retrieve new MediGrams and send outbound MediGrams. Then, the VER_MSS calls the ER_Manager to update the ER scene. The CULL and DRAW threads are then updated. Next, the VER_MSS calls the IO Managers to retrieve device input events and store them into the CODB. Finally, the VER_MSS calls the Renderer to update the scene. The GUI is updated using the current input events at the end of each frame cycle.

Upon exiting the main loop, a final aspect of the VER_MSS main is support for graceful termination. The VER_MSS function calls the *pfuExitGUI*, *pfuExitInput*, and *pfuExitUtil* clean-up functions to remove Performer data pool files created during processing. In addition, the VER_MSS executes all non-null destructors of subordinate classes, and ultimately calls *pfExit* to terminate Performer processing.

5.4.1.3 MSS Interface

The VER_MSS provides the only visible interface to the doctor trainee. As such, the VER_MSS implements a minimal interface to provide on-screen information for performance monitoring, and a GUI for useful functions. The performance monitoring is implemented as an on-screen display of the current MediGram statistics. Using a GL-overlay, this information is displayed over the scene, and is updated every simulation cycle by the DRAW thread. The display shows the current Patient Vitals MediGram received, and the most recent Doctor Treatment MediGram sent. In addition, the display contains MediGram indicators that flash to graphically indicate MediGram network activity. The standard Performer statistics display may also be rendered when required.

The VER MSS also implements a GUI bar along the bottom 4 percent of the screen, as shown in Figure 5-5. This optional panel integrates *libpfui* widgets to permit control of MSS application parameters. The GUI bar includes controls to toggle the GUI panel, terminate the MSS application, display Performer statistics, display MediGram status overlay, continuously display the currently selected object, and release the current object. The GUI is intended to provide a simple capability to control the MSS with minimal intrusion into the visible screen area. The GUI bar is implemented using the *libpfui* interface tools, and the viewing channel is partitioned and scaled to preserve scene visibility when the GUI is enabled.

Figure 5-5. VER_MSS GUI bar.

The GUI bar may not be visible when low-resolution (less than 1280 x 1024 pixel) head-mounted display devices are used. The limitations of the *libpfui* widgets do not permit widget text to be scaled. Thus, a supplemental GUI capability using a small POD panel is provided for the MSS. This POD control panel is implemented to replace the GUI bar in low-resolution configurations. The POD panel contains buttons that perform all of the primary GUI bar functions, and may be positioned and locked at any time. When locked, the POD panel follows movement of the view. The POD, unlike the *libpfui* GUI, is an in-scene element. As a result, the POD is tethered to the view at a close distance (1 centimeter).

5.4.2 MSS Support Classes

The MSS Support Classes include the MSS_Renderer and the MSS_Medigram_Manager classes. Both of these are described in this section.

5.4.2.1 MSS_Renderer

The rendering functions of the VER_MSS are divided into two class implementations: (1) General Performer configuration accomplished by the Common_Renderer, and (2) Simulation specific rendering functionality accomplished by the MSS_Renderer. Figure 5-6 shows the implemented structure of the Renderer.

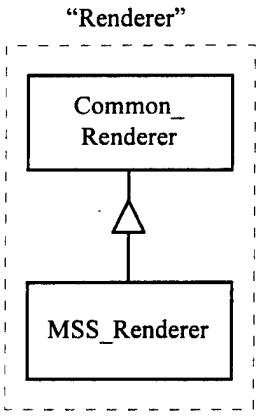


Figure 5-6. MSS_Renderer class structure.

The MSS_Renderer class inherits from the Common_Renderer class, and adds rendering functionality unique to the VER MSS. This functionality includes initialization of the view in the scene, configuring and controlling view-based collision detection, processing panic events to reset the view, controlling the view mode, and updating the view with data from the mouse or other input device.

The view is closely tied to the rendering functions accomplished by Performer. As a result, the MSS_Renderer contains algorithms for modifying and controlling the user view. The MSS_Renderer also includes motion models that permit the trainee to change the view in one of two modes. For strict simulation purposes, a "walking" mode that supports ground following and collision detection is available, using a basic terrain following algorithm. The walking mode restricts view motion to the lateral X-Y plane of the ER floor. For rapid exploration of the scene,

a "flying" mode is also available that permits the view to be modified without physical constraint.

5.4.2.2 MSS_MediGram_Manager

The MSS_MediGram_Manager class encapsulates the functions of Sheasby's Medical Network Manager class. This encapsulation approach insulates the interface to the VER_MSS from potential changes to the Medical Network Manager Interface. The functional interface is based on methods that perform the following MSS-specific functions:

1. Read the local MSS CODB.
2. Package information into correctly formatted MediGrams.
3. Broadcast the MediGram package to the PCS MediGram Manager.
4. Receive inbound MediGrams from the PCS.
5. Store formatted MediGrams in the MSS CODB.

MSS_Medigram_Manager
MSS_Medigram_Manager *~MSS_Medigram_Manager* *Initialize_Inbound* *Initialize_Outbound* *Initialize_PatientRec* *Update_Outbound* *Retrieve_Inbound* *Enable* *Disable* *Show_Patient_Vitals* *Show_Doctor_Treatment* *Show_Patient_Record* *Get_Patient_Vitals* *Get_Doctor_Treatment* *Get_Patient_InitRec*

Figure 5-7. MSS_Medigram_Manager class methods.

As shown in Figure 5-7, the MSS_MediGram_Manager contains methods to initialize, send, retrieve, and control how MediGrams are managed on behalf of the MSS. Three CODB pointers are created and maintained in the MediGram Manager: one for assembling Doctor_Treatment MediGrams, one for accepting Patient_Vitals MediGrams, and one for accepting Patient_Record MediGrams. The MSS_Medigram_Manager class sends Doctor_Treatment

MediGrams, and checks for the receipt of new Patient_Vitals and Patient_Record MediGrams using the CODB pointers. As shown, the MSS_Medigram_Manager class includes other methods to get and display MediGrams in the CODB.

5.4.3 ER_Manager class

The ER_Manager is the executive class for the synthetic ER environment. The ER layout managed by the ER_Manager is shown in Figure 5-8. The trainee is inserted into the synthetic ER, which contains static geometry objects such as walls, shelves, and the floor. Also contained within the synthetic ER are dynamic geometry objects with continuous processing requirements, such as monitors and lights.

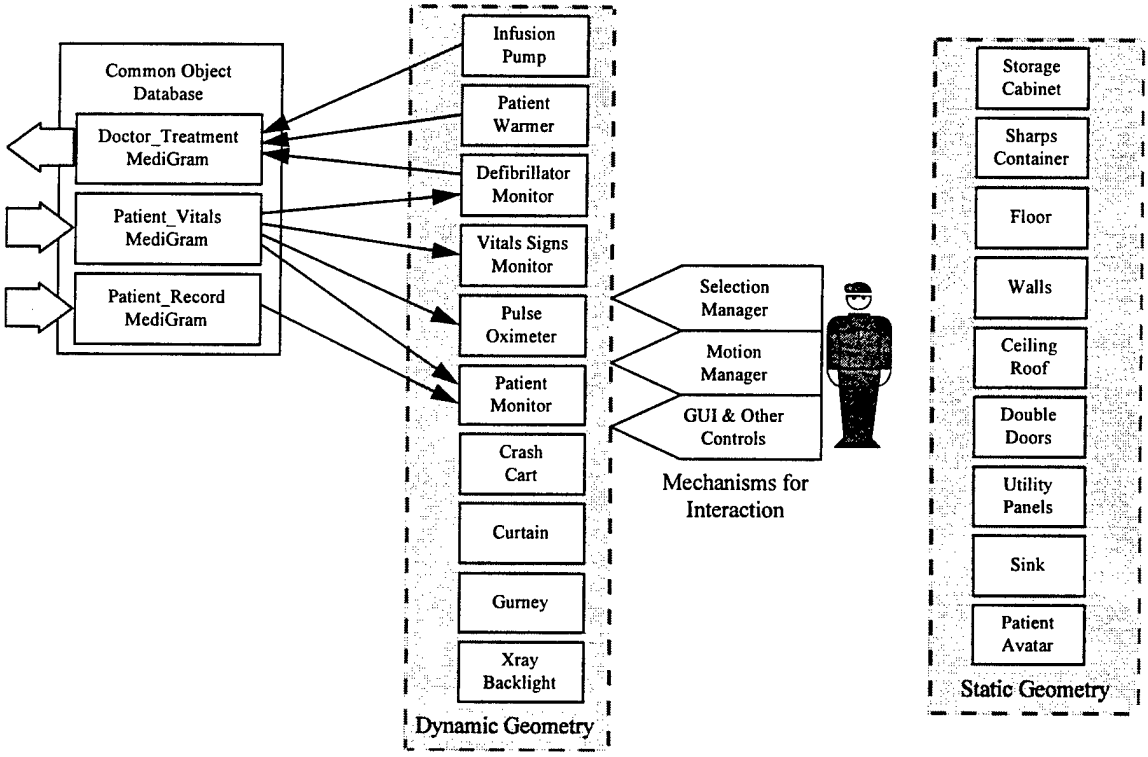


Figure 5-8. ER_Manager Implementation overview.

The ER_Manager creates and maintains the Performer scene for the synthetic ER using both object types. In addition, the ER_Manager regulates interaction with the scene by managing calls to the Selection_Manager and the Motion_Manager. This functional interaction augments the GUI generated by the VER_MSS main and provides a complete interface capability for the trainee.

The functions performed by the ER_Manager are categorized into 5 areas:

- Initialization of Apparatus Sub-tree. All dynamic and static objects are instantiated in the ER_Manager class. Thus, all apparatus and other scene elements are aggregate members of the ER_Manager class. All scene objects are grouped in an apparatus *pfGroup* structure, which is used to pass the geometry to other classes. The *pfNodeTravMask* is set to activate intersection testing on the entire apparatus group and all descendants. Instantiation of geometry is performed by calling the Model constructor for static scene objects, or the appropriate apparatus class constructor for dynamic scene objects.
- Avatar Insertion. The ER_Manager provides a mechanism to load the geometry database of a specified patient avatar. The Performer tree structure created for the avatar includes a *pfSwitch* that parents two individual Model instances: one for a dressed avatar, and one for the undressed counterpart. All geometry is loaded and stored during initialization of the MSS application, so that geometry switching does not delay the application during runtime.
- Object Selection. As part of the Scene Tree initialization, the ER_Manager creates an instance of the Selection_Manager to designate scene elements. During normal processing, the Selection_Manager is polled by the ER_Manager, so that a "current object" may be identified as required. The numeric identifier of the current object, as returned by the Selection_Manager, is stored by the ER_Manager for further processing.
- Object Movement. The ER_Manager also instantiates the Motion_Manager during initialization. During normal processing, the Motion_Manager accepts the address of the currently selected class (always a Model class). The Motion_Manager then reads mouse input from the CODB to change the position and orientation of the selected object. The ER_Manager controls operation of the Motion_Manager by setting motion flags to direct the type of motion for each apparatus object.
- Object Updates. The ER_Manager triggers updates to all apparatus objects that require per-frame updates. In addition, when the Selection_Manager identifies a current object, the state of the object is modified by the ER_Manager. The Selection_Manager results are evaluated at the lowest level *pfNode* for each object that contains component geometry. If component geometry (such as a Button) is selected, the button is toggled by calling the appropriate method in the current object. Otherwise, the state of the current geometry is evaluated. If parent geometry is selected by the Selection_Manager, the *pfHighlight* state of the complete

object is toggled. Objects that are selected are highlighted and passed to the Motion_Manager with the appropriate motion flags. Objects that are de-selected are unhighlighted, and motion control by the Motion_Manager is terminated. Beyond generating events from the Selection_Manager, the ER_Manager updates all dynamic objects every frame. Most dynamic objects support movement of geometry, process patient information retrieved from MediGrams in the CODB, or write treatment values into the CODB.

The sequence of ER_Manager processing is shown in Figure 5-9. Processing in the ER_Manager begins with initialization routines, which are invoked prior to executing the main simulation loop in the VER_MSS main function. When the VER_MSS enters the main simulation loop, the ER_Manager executes during each frame. Within each cycle, the ER_Manager polls the Selection_Manager for a current object, and processes the current object if one is identified. Processing the current object involves object identification, highlighting the geometry, and preparing the Motion_Manager to process the object for user-controlled movement.

```
Initialize MSS_Geometry Group
Initialize Selection_Manager and Motion_Manager
Insert Patient Avatar

// in main simulation loop...
Poll Selection_Manager
if object X is selected
        Get index level in Selection_Manager
        if component object
                generate event for component object in current object class
        else
                current object = X
                highlight current object
                configure Motion_Manager for current object
        end if
update Motion_Manager
update dynamic objects
```

Figure 5-9. ER_Manager processing.

The ER_Manager class methods that provide these functions are shown in Figure 5-10. The Initialize method prepares the Performer scene tree by loading all geometry databases into an apparatus group. Similarly, the Load_Patient method populates the tree with two specified patient avatar Model instances--one for the dressed avatar, and a corresponding Model for the

undressed avatar. The Update method performs the aforementioned update functions, and Get_Ground identifies the ground Model instance for floor following requirements. Finally, Toggle_Selected_Object activates or deactivates geometry highlighting for the current object, and sets the current object to an appropriate new value. The Deselect method eliminates highlighting on the current object and forces the current object to be NULL.

ER_Manager
ER_Manager *~ER_Manager* *Initialize* *Load_Patient* *Update* *Get_Ground* *Toggle_Selected_Object* *Deselect*

Figure 5-10. ER_Manager class methods.

5.4.4 Apparatus Classes

All geometry databases are imported into Performer using, as a minimum, the Model support class to translate the database into the appropriate tree structure. The Model shared class provides a core capability for converting geometry databases into performer sub-trees, and managing the sub-trees based on the requirements of the simulation. However, if the geometry database contains articulated features that must be animated in the Performer scene, this functionality must be programmed into a special apparatus class. As discussed in the design, the specialized class must properly structure and manipulate the Performer tree according to the desired functionality.

For instance, a defibrillator may be imported into the Performer scene as a static model instance using the Model class. While simple, this does not permit features of the defibrillator, such as the buttons, to move when pressed. To implement this functionality, a class to support the defibrillator is needed to properly structure the Performer tree, and provide an abstract interface that implements the motion and underlying functionality of the buttons.

As discussed in Chapter 2, information was gathered from a variety of sources, to include pictures of actual ER facilities and equipment. Beyond its usefulness for creating the 3-

D geometry databases, this information was also helpful for determining some of the functional aspects of the apparatus. This section describes the implementation of the MSS apparatus classes. As an aid in following the discussion, the Performer tree structure for the Beamlite class shows the representative Performer tree structure of an apparatus class. For brevity, Performer tree diagrams of all other apparatus classes are grouped in Appendix A.

5.4.4.1 Beamlite class

The Beamlite class simulates the functionality of overhead directional lighting, and is derived from the Model class. The Beamlite parent contains the geometry of the Beamlite housing. Attached to the parent's lower *pfDCS* node is a *pfSCS* node that hosts a *pfLightSource* node, and a *pfSwitch* node. The *pfSwitch* node, in turn, parents two additional Model instances: one for a transparent "unlit" glass panel, and one for an illuminated glass panel. Also attached to the lower *pfDCS* node is a toggle button Model object. Figure 5-11 shows the Performer tree structure for the Beamlite. Each bordered group represents an instance of the Model class.

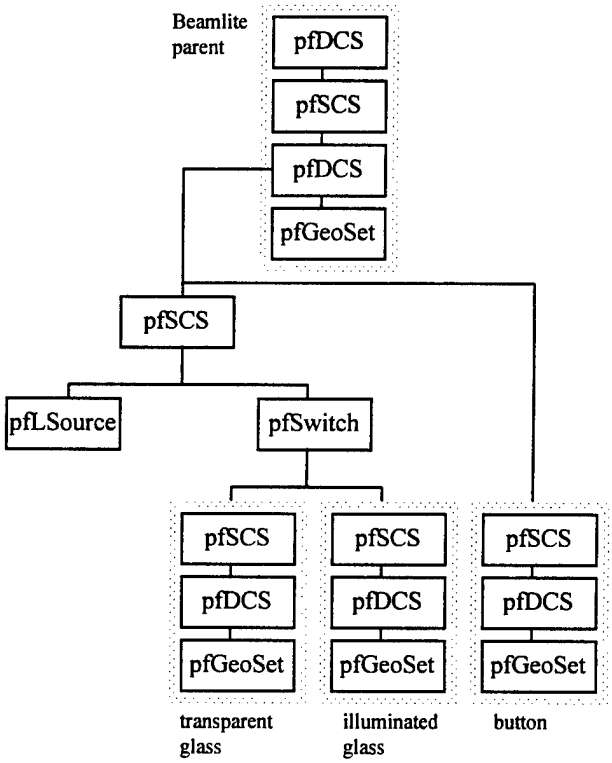


Figure 5-11. Performer tree structure for Beamlite class (others in Appendix A).

The methods provided by the Beamlite class are shown in Figure 5-12. The Configure_Light method is called from the constructor of the Beamlite object to build the aforementioned Performer scene tree. During an MSS simulation, the button activates and deactivates the Beamlite object via the Toggle_Power_Event. The *pfDCS* for the button is updated to translate the button according to whether or not it is pressed via the Update_Button method. In addition, the state of the button is used to activate the *pfSwitch* in the sub-tree using the Update method. If the button is in the "on" position, the *pfLightSource* is activated and the illuminated glass Model is displayed on the light fixture. This causes the Beamlite object to generate the effects of local lighting. If the button is in the "off" position, the *pfLightSource* is deactivated and the transparent glass Model is displayed on the light fixture.

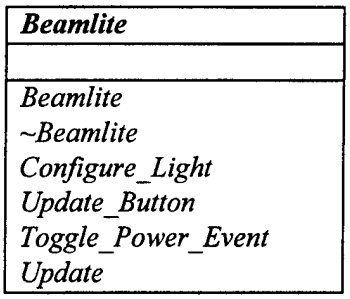


Figure 5-12. Beamlite class methods.

Because local lighting is an expensive operation in Performer, the Beamlite object is the only MSS component to use it. The *pfLightSource* node is configured to be a directional light source with a conic light dispersion. All other lighting requirements are based on infinite light sources. Local lighting produces the best results when the geometric complexity of the scene is relatively high (contains many polygons), and the materials associated with geometry are reflective [IRIS96]. Thus, the center of the MSS scene, directly over the patient avatar is an ideal location to generate local lighting effects.

5.4.4.2 Crash_Cart class

The Crash_Cart class simulates the functionality of an emergency "crash cart," and is derived from the Model class. The parent Model contains the geometry of the cart housing, without drawers or wheels. Attached to the parent's lower *pfDCS* node is a *pfGroup* node that hosts three drawer Model instances and four wheel Model instances.

The methods provided by the Crash_Cart class are shown in Figure 5-13. The Init_Crash_Cart method is called from the constructor of the Crash_Cart object to build the Performer scene tree. During an MSS simulation, the Update method updates the *pfDCS* nodes of each of the wheels to rotate them according to the specified direction of motion. This feature is used to make movement of the Crash_Cart object appear natural.

Crash_Cart
Crash_Cart *~Crash_Cart* *Init_Crash_Cart* *Toggle_Drawer_Event* *Update*

Figure 5-13. Crash_Cart class methods.

In addition, each of the drawer Models may be moved about their *pfDCS* nodes to simulate opening and closing them. The motion of the drawers is controlled via the Toggle_Drawer_Event method. An internal event processor queues open- and close-drawer events, so that events may be processed over several rendering frames in small incremental translations. This produces a smooth drawer motion that is visually superior to an instantaneous drawer update.

5.4.4.3 Curtain class

The Curtain class simulates the functionality of an emergency room curtain, and is unique in that it is the only object class not derived from the Model class. The reason for this implementation decision is based on the approach taken to implement the curtain. To provide an open-and-close capability, a single section of curtain is cloned 15 times using the *pfClone* operation. The cloned curtain segments are then added or removed from the Curtain object subtree to simulate changes in curtain configuration.

The unique structure of the Curtain object is primarily based on creating a *pfGroup* node to parent a Model of a single instance of the curtain segment geometry. This single segment Model is referenced 15 times using *pfClone* nodes, which are also attached to the parent *pfGroup*. The curtain segment geometry is modeled to permit the instances of the model to be joined without a visible seam. The *pfClone* node permits existing geometry to be instanced

without wasting memory. Because the curtain is repeated without visual penalty, use of the *pfClone* reduces the curtain geometry from over 1500 polygons to about 100.

The methods of the Curtain class are shown in Figure 5-14. The Init_Curtain method is called from the constructor of the Curtain object to build the Performer scene tree. During an MSS simulation, the motion of the curtain is controlled via the Toggle_Curtain_Event method. This method uses an internal event processor to extend the modification of the curtain sub-tree over several frame updates. In this manner, the curtain geometry is modified over a definite time interval that better approximates the motion of a curtain. The initial curtain segment is always visible. Closing the curtain entails sequentially adding *pfClone* instances to the parent *pfGroup* via the Enable_Segment method. Opening the curtain entails sequentially removing *pfClone* instances of the curtain segment from the parent *pfGroup* with the Disable_Segment method.

Curtain
Curtain *~Curtain* *Init_Curtain* *Disable_Segment* *Enable_Segment* *Toggle_Curtain_Event* *Update*

Figure 5-14. Curtain class methods.

5.4.4.4 Dinamap class

The Dinamap class simulates the functionality of a "Dinamap" dedicated patient vital signs monitor, and is derived from the Model class. The parent Model contains the apparatus housing, without buttons or face plate. Attached to the parent's lower *pfDCS* node is a *pfGroup* node that hosts a control backpanel Model object and ten button-switch Model objects, located on the control backpanel.

The methods provided by the Dinamap class are shown in Figure 5-15. The Init_Dinamap_Buttons method is called from the constructor of the Dinamap object to build the Performer scene tree. During an MSS simulation, each button-switch is toggled on and off using the Toggle_Button_Event method. This causes the Update_Button method to modify the *pfDCS* for each pressed button-switch by a rotation of 60 degrees above or below the neutral centerline

axis. The control back-panel does not move, and acts as a canvas upon which to draw display updates.

Dinamap
Dinamap *~Dinamap* *Init_Dinamap_Buttons* *Update_Button* *Update_Display* *Toggle_Button_Event* *Draw_Settings*

Figure 5-15. Dinamap class methods.

Display updates are generated by a *pfNodeTravFuncs* method, Draw_Settings, which draws the LCD readouts on the face of the monitor after the DRAW thread completes rendering of the control back-panel. These values are updated by calling the Update_Display method to access the CODB for current Patient_Vitals readings. The information displayed on the control back-panel includes arterial pressure, systolic blood pressure, diastolic blood pressure, and heart rate. The GL linefont library provides the drawing routines needed to display these values. The data displayed on the control back-panel is obtained by reading the most current Patient Vitals record from the CODB, formatting it, and generating GL commands to render it each frame.

5.4.4.5 Gurney class

The Gurney is almost a completely static class, derived from the Model class. The parent Model contains the geometry of the mattress and supporting structure without wheels. Attached to the parent's lower *pfDCS* is a *pfGroup* that parents four wheel Model instances. The methods provided by the Gurney class are shown in Figure 5-16.

Gurney
Gurney *~Gurney* *Init_Gurney* *Update*

Figure 5-16. Gurney class methods.

The Init_Gurney method is called from the constructor of the Gurney object to build the Performer scene tree. During an MSS simulation, the Update method causes each of the wheels may be rotated about their respective *pfDCS* nodes according to a specified orientation. This feature is used to make movement of the Gurney object appear natural.

5.4.4.6 Infusion_Pump class

The Infusion_Pump class simulates the functionality of a rapid infusion pump, and is derived from the Model class. The parent Model contains the apparatus geometry housing without the wheels or control panel instrumentation. Attached to the parent's lower *pfDCS* node is a *pfGroup* node that hosts a control back-panel Model, four wheel Models, and eight button Models.

The methods provided by the Infusion_Pump class are shown in Figure 5-17. The Init_Infusion_Pump method is called from the constructor of the Infusion_Pump object to build the Performer scene tree. During an MSS simulation, each of the wheel Model objects is rotated about their respective *pfDCS* nodes via the Update_Wheels method. This feature makes movement of the Infusion_Pump object appear natural. The control back-panel acts as a canvas upon which to draw the buttons and current configuration settings of the Infusion_Pump. The flow rate value is rendered by a *pfNodeTravFuncs* method, Draw_Settings, that draws the current flow rate value on the control back-panel after the DRAW thread finishes rendering the control back-panel geometry.

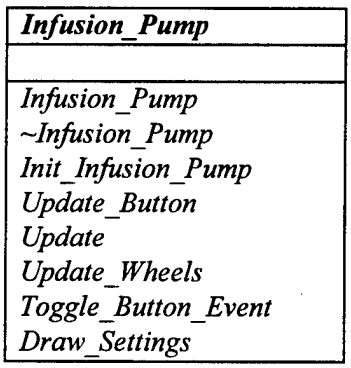


Figure 5-17. Infusion_Pump class methods.

The Infusion_Pump object has a functional control panel, and any of the button Models may be pressed. A button press event generated by the Toggle_Button_Event method causes the

Update_Button method to translate the appropriate button geometry by updating the respective *pfDCS* matrix. Of the eight button Models, four are radio button Models that represent substance categories, two are toggle buttons for activating pump power and patient connection, and two are stateless push-buttons that permit the pump flow rate to be increased and decreased.

The Update method regularly updates the CODB, even when no IV treatment is being administered. However, when a specified Power toggle button, Connect toggle button, and substance radio button are pressed, the Doctor Treatment MediGrams generated from the MSS include Infusion_Pump treatment data. The Update method restricts the minimum and maximum flow rates permissible, so treatments will always be subject to a finite flow rate.

5.4.4.7 Oximeter class

The Oximeter class simulates the functionality of a pulse oximeter monitor, and is derived from the Model class. The parent Model contains the apparatus geometry housing without control panel objects. Attached to the parent's lower *pfDCS* node is a *pfGroup* node that hosts a control back-panel Model object and two button Model objects, located on the control back-panel.

The methods provided by the Oximeter class are shown in Figure 5-18. The Init_Oximeter_Buttons method is called from the constructor of the Oximeter object to build the Performer scene tree. During an MSS simulation, each button may be toggled on and off using the Toggle_Button_Event method. This causes the Update_Button method to modify the *pfDCS* node for each pressed button by a rotation of 40 degrees above or below the centerline. The control back-panel does not move, and acts as a canvas upon which to draw digital readout updates.

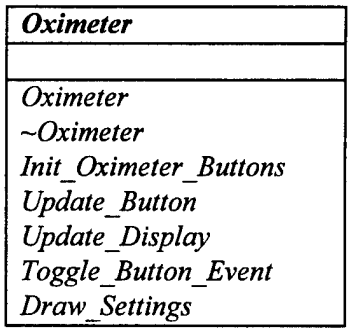


Figure 5-18. Oximeter class methods.

Instrument updates are generated by a *pfNodeTravFuncs* method, Draw_Settings, that draws the LCD readouts on the control back-panel face after the DRAW thread finishes rendering the control back-panel geometry. These values are updated by calling the Update_Display method to access the CODB for current Patient_Vitals readings. The information displayed on the control back-panel includes patient blood oxygen and heart rate. These values are drawn using GL linefont library calls. The data that is displayed on the control back-panel is obtained from the most current Patient_Vitals MediGram in the CODB.

5.4.4.8 Patient_Warmer class

The Patient_Warmer class simulates the functionality of a patient warming system, and is derived from the Model class. The parent Model contains the complete warmer housing without control panel instrumentation. Attached to the parent's lower *pfDCS* node is a *pfGroup* node that hosts a control back-panel Model and four button Models.

The methods provided by the Patient_Warmer class are shown in Figure 5-19. The Init_Patient_Warmer method is called from the constructor of the Patient_Warmer object to build the Performer scene tree. The Patient_Warmer has a functional control panel. Any of the button Models may be pressed. A button press generates a call to the Toggle_Button_Event method, which in turn activates the Update_Button method. This method translates the appropriate button by updating the corresponding *pfDCS* matrix. Of the four button Models, two are toggle buttons for activating power and patient connection, and two are stateless push-buttons that permit the temperature to be increased and decreased.

Patient_Warmer
Patient_Warmer *~Patient_Warmer* *Init_Patient_Warmer* *Update_Button* *Update* *Toggle_Button_Event* *Draw_Settings*

Figure 5-19. Patient_Warmer class methods.

The control back-panel is used as a canvas upon which to draw the buttons and current configuration settings of the Patient_Warmer. The temperature value is rendered by a

pfNodeTravFuncs method, Draw_Settings, that draws the current temperature value on the control back-panel after the DRAW thread renders the control back-panel geometry. The push buttons may be used to modify the value displayed on the control back-panel.

The Update method regularly updates the CODB, even when no heat treatment is being administered. However, when the specified Power toggle button and Connect toggle button are pressed, Doctor Treatment MediGrams generated from the MSS include Patient_Warmer treatment inputs. Thus, the Update method uses current state information to control how the CODB is updated for patient warmth treatments. In addition, the Update method restricts the minimum and maximum temperature permissible, so treatments will always be within a finite range.

5.4.4.9 Xraylite class

The Xraylite class simulates the functionality of an x-ray viewing back light, and is derived from the Model class. The parent Model contains the geometry of the unit housing, with none of the lighting panel or control panel geometry. Attached to the parent's lower *pfDCS* node is a *pfGroup* node that hosts four *pfSwitch* nodes, and four button Models. Attached to each *pfSwitch* node is a lit panel Model and an unlit panel Model.

The methods provided by the Xraylite class are shown in Figure 5-20. The Init_Xraylite method is called from the constructor of the Xraylite object to build the Performer sub-tree. During an MSS simulation, each of the back light panels may be turned on and off by pressing a corresponding button. This causes a call to Toggle_Lite_Event, to display the correct light geometry.

Xraylite
Xraylite *~Xraylite* *Init_Xraylite* *Update_Button* *Toggle_Lite_Event*

Figure 5-20. Xraylite class methods.

Pressing a button also generates a call to the Update_Button method update the *pfDCS* matrix of the button. Each button corresponds to the control of an individual panel. When the

button for a panel is pressed, the state of the button is used to activate the appropriate child of the corresponding *pfSwitch* node. Thus, when a power-on event is detected, the *pfSwitch* activates the lit panel child Model. Similarly, when a power-off event is detected, the *pfSwitch* activates the unlit panel child Model.

Although the Scene-tree complexity of the Xraylite is relatively high when compared to the other geometry classes, the underlying processing requirements are simple. There are no *pfTravNodeFuncs* calls to execute, and no CODB access requirements.

5.4.4.10 Defibrillator class

The Defibrillator class simulates the functionality of a combined defibrillator and patient monitor, and is derived from the Model class. The parent Model contains the apparatus housing without control panel geometry. Attached to the parent's lower *pfDCS* node is a *pfGroup* node that hosts a control back-panel Model and ten button Models, located on the control back-panel.

The methods provided by the Defibrillator class are shown in Figure 5-21. The Init_Defibrillator_Buttons method is called from the constructor of the Defibrillator object to build the Performer scene tree. During an MSS simulation, each button-switch is toggled on and off via the Toggle_Button_Event method. This causes the Update_Button method to modify the *pfDCS* matrix for each pressed button-switch, to translate the geometry in or out relative to the control back-panel. All buttons retain their state, except for those assigned to modify the energy level. The control back-panel does not move, and is used as a canvas upon which to draw display updates.

Defibrillator
Defibrillator *~Defibrillator* *Init_Defibrillator_Buttons* *Update_Button* *Update_Display* *Toggle_Button_Event* *Draw_Settings*

Figure 5-21. Defibrillator class methods.

Display updates are generated by a *pfNodeTravFuncs* method, Draw_Settings, that draws the LCD readouts on the face of the monitor after the DRAW thread renders the control back-

panel. These values are updated by calling the Update_Display method to access the CODB for current Patient_Vitals readings, and by reading interval state variables modified by the button control panel. The information displayed on the control back-panel includes patient heart rate and a dynamic wave form that corresponds to the cardiac electrical rhythm of the virtual patient. In addition, the current energy level selected for defibrillation treatment is also displayed. All front-face displays are rendered using the GL linefont library and the Pulse class (discussed next). The heart rate and wave form data displayed on the control back-panel is obtained by reading the most current Patient Vitals record from the CODB.

The Update method regularly updates the CODB, even when no defibrillation treatment is being administered. However, when the Power toggle button and Charge toggle button are pressed, the Doctor Treatment MediGrams generated from the MSS include Defibrillation treatment data. Thus, the Update method uses current state information to control how the CODB is updated for defibrillation treatments. In addition, the Update method restricts the minimum and maximum defibrillation energy levels permissible, so treatments will always be subject to a finite energy level rate.

5.4.4.10.1 Pulse and GL_Canvas classes

The Electrocardiogram (ECG) monitor integrated into the defibrillator displays a wave form that corresponds to the electrical signals of the patient's cardiac sinus rhythm. Because the activity of the patient's heart is dynamic, so is the ECG display. Both the tempo and shape of the wave form can vary in response to the patient's physiological status.

The implementation of the wave form is an important consideration in the implementation of the Defibrillator class. In particular, the mechanism to display a dynamic wave form pattern must be carefully balanced to accommodate (1) the speed at which the data display may be rendered, (2) the fidelity of the data display, and (3) the subjective degree to which the data display will vary over time.

A first attempt to implement cardiac sinus wave forms was to derive a mathematical formula that permits direct computation of the electrical amplitude for any time interval in a typical wave form pattern. However, this ultimately proved to be too difficult due to the time and complexity associated with computing a formula in every frame. Thus, a lookup table approach is implemented to define a representative wave form over discrete time intervals. An approximation of a continuous wave form is generated by stepping through a lookup table to

trace the representative wave form according to the required frequency of the display. This approach eliminates the need for a precise mathematical definition of the cardiac rhythms, thereby minimizing computational overhead in the DRAW thread.

The Pulse class supports the dynamic display requirements of the Defibrillator by providing a collection of operations for dynamically displaying wave form patterns using the lookup table strategy. The Pulse class is based on the GL_Canvas virtual class, which provides the functionality necessary to establish a definable GL display area. The GL_Canvas uses IRIS GL calls, and gets called from within a *pfNodeTravFuncs* call.

The Pulse class is used to update a display buffer with signal information taken from one or more wave form definitions (stored as an array of triples). The Pulse class is always in one of two modes: cycling or idle. These states affect how the contents of the active display buffer are modified. During frames between pulse events (or cardiac failure situations), the Pulse class is idle and a zero "flat-line" value is stored at the head of a continuously updated display buffer. However, when a signal must be rendered, the display buffer is updated by tracing through the current wave form pattern. The updates involve traversing the pattern in step with the next pixel location in the displayable area, and copying the coordinates of the wave form into the display buffer. The signal trail is "constructed" every frame by walking through the signal pattern. As depicted in Figure 5-22, a continuous signal display alternates between "idle" and "cycle" states.

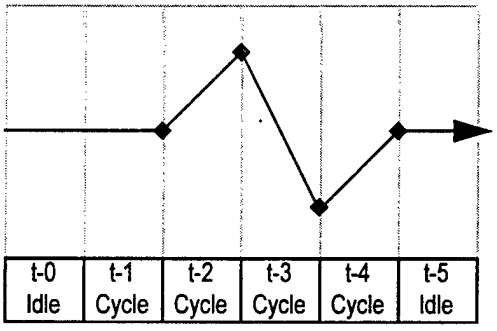


Figure 5-22. Dynamic wave form construction in the Pulse class.

The Pulse class provides real-time wave form updates, using system time produced by *pfGetTime*. The heart rate is specified in beats per minute by the Defibrillator class. This value is then converted into a pulse time interval in Hertz using Equation 5-1. Calls to *pfGetTime* are used each frame to determine if the computed time interval has passed, and if so, the Pulse class begins another cycle.

$$Pulse\_Interval = \frac{1}{\left(\frac{\frac{HR \text{ beats}}{\text{minute}}}{\frac{60 \text{ seconds}}{\text{minute}}}\right)} = \frac{60}{HR} \quad (5\text{-}1)$$

To provide a graphically correct display, the high-persistence effect common to CRT monitors is simulated by the Pulse class. Thus, the signal is brightest near the leading edge, and weakens progressively in the tail section. This effect is achieved using a color ramp that contains various intensities of a primary color (such as green) in discrete intervals over 8 bits (in the range 0 to 255). The size of the color ramp is directly proportional to the length of the signal tail, so the longer tails have smoother color ramps. The color ramp is stored in an array such that the highest signal color intensity (255) is stored at the initial array address and zero-intensity (black) is stored at the final array address. To obtain the desired effect, the signal must be displayed over a black background.

The Pulse class also provides signal wrapping within the display area defined in the GL_Canvas class. This is implemented by using an internal double buffer to store the current signal trail. When the signal wraps around the screen, a secondary buffer is temporarily used to write an updated pulse on the left edge of the displayable area while the previous trail from the primary buffer on the right edge fades. When the trail remnant has fully propagated beyond the edge of the display area, the contents of the secondary buffer are copied into the primary buffer and the process starts anew.

5.4.4.10.2 Pulse Wave form Implementation

To provide useful cardiac rhythm displays, a wave form for normal cardiac rhythm was originally sampled over 40 points to capture the essential traits of the curve, as shown in Figure 5-23.

This approach provided an immediate solution for the Pulse class with acceptable visual results. However, the solution produced a noticeable rendering lag, caused by a wave form definition too big to render quickly. The display of a heart beating at 150 beats per minute could not be generated, because approximately 30 percent of each wave form was truncated by the next wave form cycle.

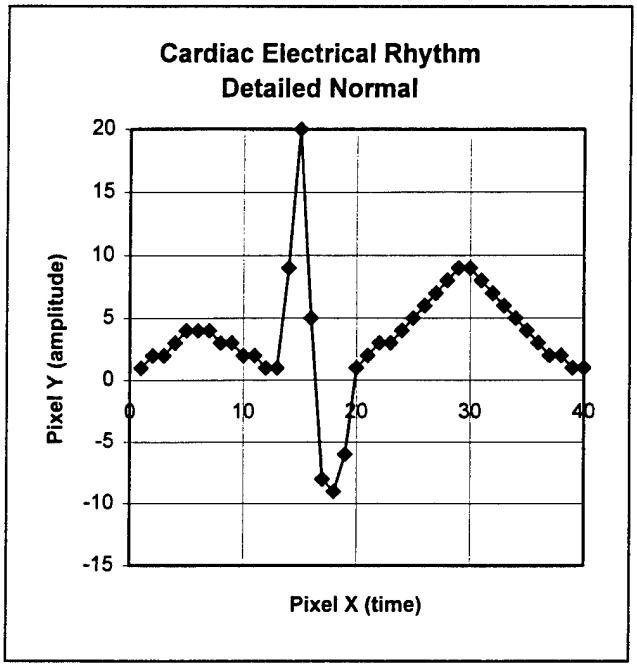


Figure 5-23. Original wave form definition.

A modification to this strategy minimizes the size of the wave form to permit faster rendering. The typical cardiac rhythm curve used in the original approach was sampled at about half of the frequency to produce a smaller pattern. Wave form patterns for a regular and traumatic heart rhythm are shown in Figure 5-24 and Figure 5-25 respectively. This modified strategy provides acceptable results at higher rendering speeds, because minimal burden is placed on the DRAW thread every frame. Based on empirical testing results, the MSS_Renderer is capable of generating a continuous wave form that exceeds 200 beats per minute for wave forms that are defined with no more than 20 sample points, and still faster for smaller wave forms. An important feature of rendering the wave forms is to switch wave forms to accommodate the heart rate. Using this strategy, new wave forms may be defined for a variety of cardiac stress situations in a short time.

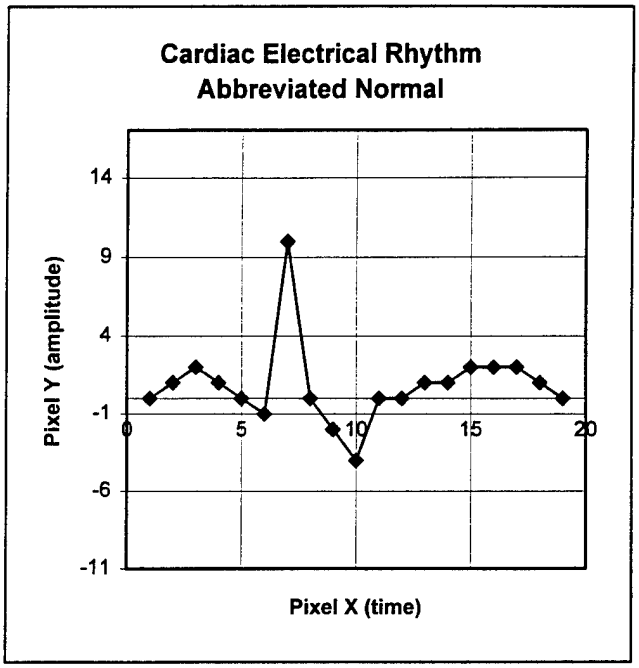


Figure 5-24. Final wave form definition for normal heart rhythm.

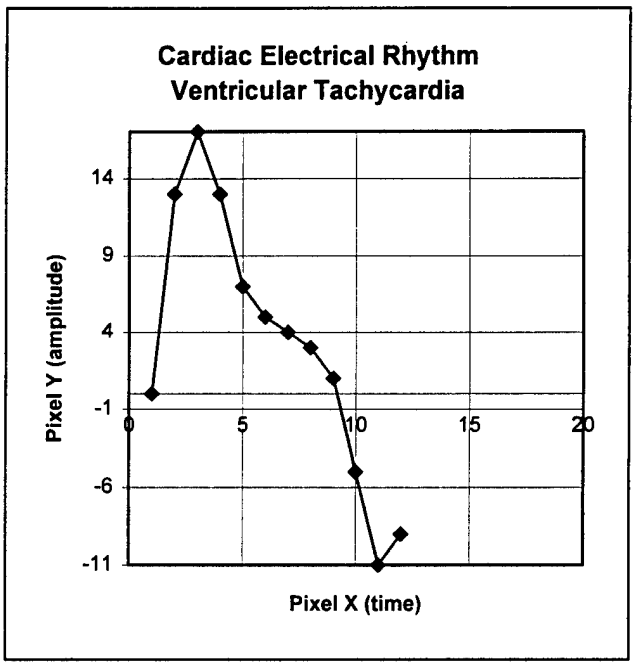


Figure 5-25. Final wave form definition for cardiac arrhythmia.

5.4.4.11 Primary Monitor

The Monitor class simulates the functionality of a dedicated ER patient monitor, and is derived from the Model class. The parent Model contains the apparatus housing without control panel geometry. Attached to the parent's lower *pfDCS* node is a *pfGroup* node that hosts a control back-panel Model and twelve button Models, located on the control back-panel.

The methods provided by the Monitor class are shown in Figure 5-26. The Init_Monitor_Buttons method is called from the constructor of the Monitor object to build the Performer scene tree. During an MSS simulation, each button-switch is toggled on and off via the Toggle_Button_Event method. This causes the Update_Button method to modify the *pfDCS* matrix for each pressed button-switch, to translate the geometry in or out relative to the control back-panel. All buttons retain their state. The control back-panel does not move, and is used as a canvas upon which to draw display updates.

Monitor
Monitor *~Monitor* *Init_Monitor_Buttons* *Update_Button* *Update_Display* *Toggle_Button_Event* *Draw_Settings*

Figure 5-26. Monitor class methods.

Display updates are generated by a *pfNodeTravFuncs* method, Draw_Settings, that draws the LCD readouts on the face of the Monitor after the DRAW thread renders the control back-panel. These values are updated by calling the Update_Display method to access the CODB for current Patient_Vitals readings, and by reading interval state variables modified by the button control panel. The information displayed on the control back-panel includes patient heart rate, the ECG wave form displayed on the Defibrillator object, the patient's name, gender, age, blood type, temperature, systolic and diastolic pressures, and other ancillary details. Dynamic wave form updates are accomplished with the same mechanisms discussed for the Defibrillator.

5.4.5 Interaction Control

The human-computer interface of the MSS is largely based on the implementation of the Selection_Manager class to select objects, and the Motion_Manager class to move selected objects. The implementation details of these classes are discussed in this section.

5.4.5.1 Selection_Manager

Picking is a particularly useful mechanism for interacting with virtual worlds, because the notion of picking objects with the mouse is intuitive and natural [VINC95]. The purpose of the Selection_Manager is to automate the use of the Performer *pfChanPick* function to "choose" elements of the visible part of the scene in the current viewing frustum. This is done by identifying specially named nodes in the Performer Scene Tree so that they can later be identified using an in-scene picking mechanism. The picking mechanism accepts a mouse event to direct an interrogation segment into the visible scene, and returns the unique identifier of the first scene element encountered by the segment. The methods included in the Selection_Manager class to accomplish this are shown in Figure 5-27.

Selection_Manager
Selection_Manager *~Selection_Manager* *Set_Mouse_Pick_Buttons* *Get_Buttons_Used_To_Pick* *Set_Selection_Level* *Get_Level_Index* *Get_Normalized_Mouse_Coords* *Make_Path_String* *Poll* *Process_Pick* *Set_Toggle_Key*

Figure 5-27. Selection_Manager class methods.

5.4.5.1.1 Configuration

Configuration of the Selection_Manager requires that the Performer Scene Tree be properly organized. The specific configuration requirements include assigning a key-string to

each *pfNode* to be identified, defining intersection masks on pickable nodes, and configuring the mouse.

The Selection_Manager functions properly only if it can uniquely identify pickable nodes. In order to use the Selection_Manager, the Performer Scene Tree is organized according to the following rules:

- A unique 5-digit identifier is assigned to each "pickable" element of the scene. This number identifies a selected object.
- A special, predetermined key-string is used to name all pickable nodes. For every *pfNode,* there is a corresponding *pfNodeName* composed of the concatenation of the key-string and the unique identifier:

KEYSTRINGNNNNN

The KEYSTRING identifier is consistently used as a prefix to the name of all pickable nodes, and NNNNN are the decimal digits that uniquely identify the *pfNode* scene element. The key-string is provided to the Selection_Manager by the requesting agent as a parameter to the Selection_Manager constructor.

In addition to uniquely naming each pickable node in the Performer tree, each *pfNode* in the Performer tree must have intersection testing activated. This is done by setting the *pfNodeTravMask* for pickable nodes in the pickable sub-tree to a predetermined non-zero intersection mask value. Although the value of the masks vary between applications, the masks must be set for the Selection_Manager to function properly.

A final configuration requirement is defining the mouse events to activate the Selection_Manager. The Selection_Manager includes the Set_Mouse_Pick_Buttons method to define the mouse control buttons. The mouse buttons may include one or combinations of several buttons, and are queried using the Get_Buttons_Used_To_Pick method. A toggle key for activating the Selection_Manager mouse events may also be defined using the Set_Toggle_Key method.

5.4.5.1.2 Selection_Manager Processing

Once configured, the Selection_Manager is polled using the Poll method to identify scene elements designated by the mouse. The polling mechanism consists of two processing components: eye-to-scene intersection testing, and results interpretation.

The Selection_Manager initiates processing by mapping normalized mouse coordinates on the screen into directed intersection rays represented by *pfSegs*. The normalized coordinates are computed using the Get_Normalized_Mouse_Coords method. When polled, the Selection_Manager determines if in-scene picking is activated. If activated, a mouse click event is captured to process an in-scene picking operation. Additionally, an interrogation segment (*pfSeg*) is created to form a ray from the eye, through the mouse coordinates, and into the scene. The Selection_Manager may optionally use interrogation segments defined by other interface elements. This overloaded capability was added to the Selection_Manager by Capt Terry Adams to support *pfSegs* already created for use in POD-based applications, such as the AFIT Virtual Cockpit [ADAM96].

When the interrogation *pfSeg* is defined, the picking operation employs the *pfChanPick* function to intersect the visible portion of the Performer Scene tree with the interrogation *pfSeg*. The *pfChanPick* call generates a *pfNodeIsectSegs* call, that attempts to intersect the interrogation segment with the scene. If any scene elements are intersected by the interrogation segment, the results of the intersection are stored in a special *pfHits* data structure. Figure 5-28 illustrates the general process.

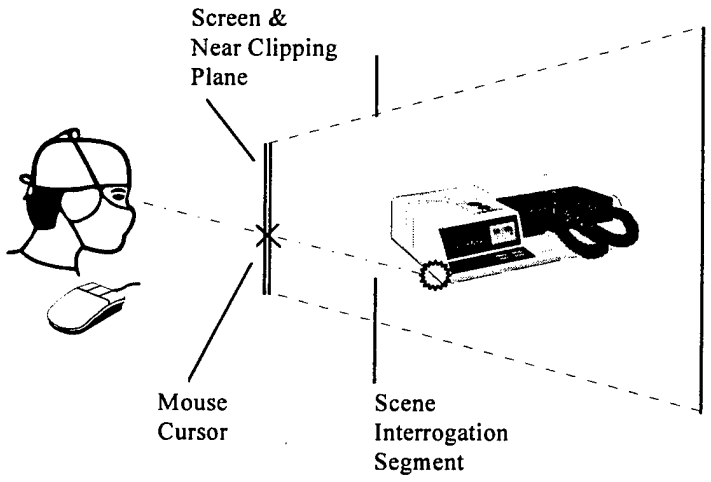


Figure 5-28. Picking using the Selection_Manager.

The extent of intersection testing may be implicitly controlled. The Selection_Manager accepts the root node of the portion of the Performer Scene Tree from which to evaluate picking requests. Thus, intersection testing is only performed from the root of the specified parent

pfNode in the Performer Scene Tree down through the descendant nodes. This permits picking interrogations to be ignored for portions of the Performer Scene Tree that should not be checked (such as the walls and floor). The benefits of limiting the search are dependent on the organization of the Performer scene tree.

After the *pfChanPick* operation completes, the Poll method evaluates the *pfHits* data structure. The path name of an intersected *pfNode*, if one exists, is stored in the *pfHits* structure. If no intersections occurred, processing the poll request terminates. A positive intersection, however, requires that the object intersected by the interrogation segment be identified using the Process_Pick method. Using a call to *pfQueryHits*, Process_Pick extracts the path name of the intersected node from the data structure. Additional processing is performed to convert the *pfNodeName* into a character string, using a call to the Make_Path_String method. The final processing requirement is to extract the unique 5-digit number for the picked element, if the node is a pickable element. Process_Pick searches the intersected node from left to right for the key-string initially provided to the Selection_Manager. The left to right search causes the key-string search to progress so that parent node names are encountered before child node names.

If the key-string is not found in the scene tree path name, Process_Pick returns a -1 result to indicate that a pickable object is not selected. If the key-string is found, however, the digits immediately following the key-string are extracted from the path name, converted into an integer, and returned as the identifier of the selected object.

An important feature of the Selection_Manager is the ability to interrogate the Performer Scene tree for several instances of the key string. A controlled search capability is provided by specifying the desired selection level in Set_Selection_Level method. This is particularly useful for identifying individual components on a selectable object, such as the buttons of a defibrillator. The ability to search for various levels of the key-string provides a corresponding capability to discern the selection of articulated features from their parent nodes. The current selection level is available through a call to the Get_Level_Index method.

Figure 5-29 shows how various levels of search are processed by the Selection_Manager. The same left-to-right traversal of the Scene Tree used to check for the initial key string may thus be reused for an arbitrary number of selection levels in the search. In cases when a Selection_Manager poll generates an intersection with a scene element at the desired level, the

numeric identifier of the object (a short integer) is returned as a result. Otherwise, if the key string at a desired level is not found, a -1 result is returned.

The Selection_Manager permits efficient selection of scene objects within a specified Performer scene sub-tree. The results are primarily attributed to the use of the CULL thread by Performer to process intersection tests of the *pfChanPick* and underlying *pfNodeIsectSegs* calls. Thus, in a multi-threaded environment, the intersection overhead is negligible even in complex scenes.

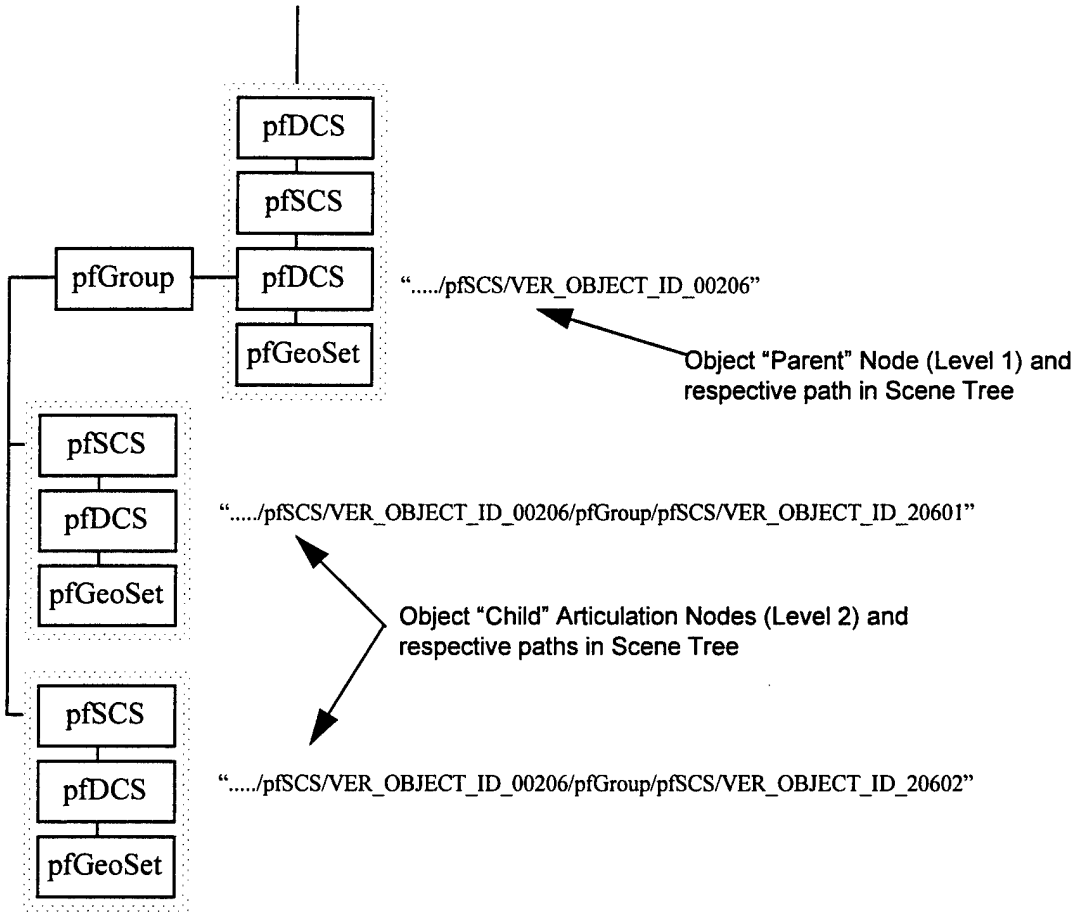


Figure 5-29. Interpreting Selection_Manager results.

5.4.5.2 Motion_Manager

The Motion_Manager is a customized motion control class, developed to meet the basic needs of the VER MSS. The Motion_Manager controls a Model class object, so the address of the currently selected object is passed to the Motion_Manager.

The methods provided by the Motion_Manager class are shown in Figure 5-30. The interface is intended to provide a basic capability for movement in five degrees of freedom. The Motion_Manager is activated for a specific Model instance using the Set_Selected_Model method. When the Motion_Manager class controls an object, the mouse structure in the CODB is evaluated every frame using calls to the Update and Get_Mouse methods. When a mouse button press event is detected, the event is recorded by the Motion_Manager. Motion is then activated if the mouse is "dragged" across the screen. The button pressed dictates the type of translation or rotation. The movement of the mouse from the original button-press event controls acceleration of motion.

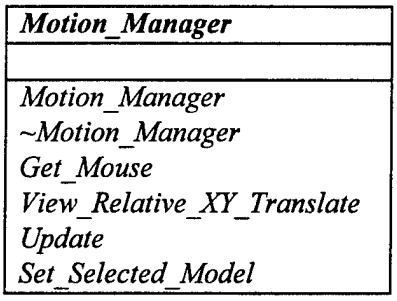


Figure 5-30. Motion_Manager class methods.

The range of motion supported by the Motion_Manager includes movement about the X, Y, and Z axes, and rotation about the heading and pitch axes of the object. Motion is controlled using the mouse to move a selected object, using the button assignments shown in Figure 5-31.

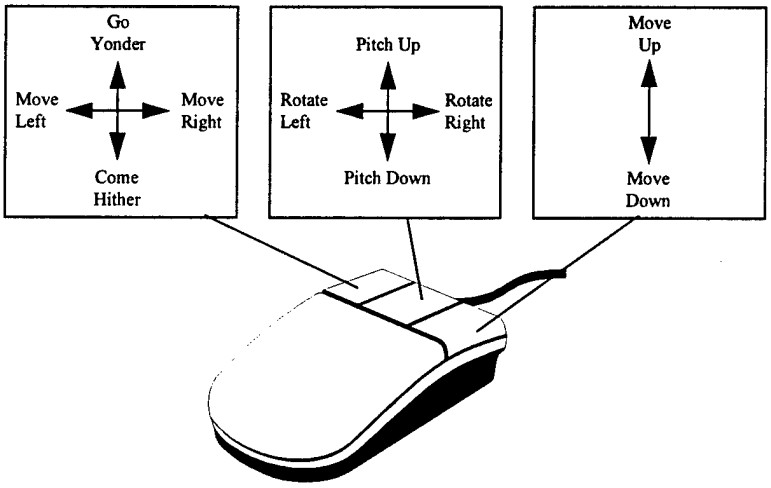


Figure 5-31. Motion_Manager mouse button assignments.

Motion is implemented relative to the current view perspective, so intuitive control of scene elements is possible from any vantage point. The computation for viewer-relative lateral movement in the X-Y plane is performed by the View_Relative_XY_Translate method, and requires that the a progression of transformation matrices be computed to perform the following steps:

1. Move object to origin with respect to the current view position and orientation.
2. Update the transformation matrix of object with updated movement data.
3. Move object from view-relative origin back to original orientation.

Thus, the calculation is of the form shown in Equation 5-2.

$$\left[\ Object\_Matrix'\ \right] = \left[\ Object\_Matrix\ \right]\left[\ Current\_View\ \right]^{-1}\left[\ \Delta_{xy}\ \right]\left[\ Current\_View\ \right] \qquad (5\text{-}2)$$

Rotation for heading and pitch movement, and movement along the z-axis are performed directly on the *pfDCS* node of the current object. These movements are not processed as viewer relative, and are less expensive to compute as shown in Equations 5-3, 5-4, and 5-5.

$$h' = h + \Delta h\text{, where } \Delta h = (curr\_mouse - last\_mouse) * rotation\_speed_h \qquad (5\text{-}3)$$

$$p' = p + \Delta p\text{, where } \Delta p = (curr\_mouse - last\_mouse) * rotation\_speed_p \qquad (5\text{-}4)$$

$$z' = z + \Delta z\text{, where } \Delta z = (curr\_mouse - last\_mouse) * rotation\_speed_z \qquad (5\text{-}5)$$

To control the motion properties of various scene elements, the Motion_Manager permits certain motion types to be selectively enabled and disabled via motion control flags. The first flag, designated as mmm_XYH, permits lateral motion in the X-Y plane, and rotation about the heading axis of the object. The other motion mode is designated as mmm_ZP, and permits motion along the Z-axis and rotation about the pitch axis of the object. Modes may be bit-wise OR-ed together. Thus, an object may have no motion mode (making it immovable), the mmm_XYH motion mode for simple ground following motion, or the bit-wise OR of mmm_XYH and mmm_ZP, which permits a full range of motion in 5 degrees of freedom.

A useful feature of the Motion_Manger is the computation of the direction of motion. This value, which is computed between position updates, is made available so that articulations (such as wheels) of the selected object may be updated. The direction of travel in the X-Y plane is used as the basis for the computation shown in Equation 5-6.

$$\text{Direction of Travel} = \text{ArcTan}\left(\frac{\Delta y}{\Delta x}\right) \qquad (5\text{-}6)$$

Another important implementation feature of the Motion_Manager is the limited control it places on the Performer sub-trees of the object that it moves. The Motion Manager is a shared resource in the VER MSS. As such, it does not take control of a single scene element. Rather, it relies heavily on the data structure and methods of the individual Model class to accomplish the work associated with updating *pfDCS* matrices. In this same way, additional functionality such as a collision detection mechanism may be added to prevent scene elements from overlapping.

5.5 PCS Implementation

The PCS implementation takes advantage of extensive reuse. The PCS relies on the Common_Renderer, IO_Mouse, IO_Keyboard, CODB, Model, and Selection_Manager classes. These classes do not require further explanation. Components unique to the PCS include the VER_PCS main function, the PCS_Medigram_Manager, and the Script_Manager. This section describes how these classes are implemented and integrated into the PCS.

5.5.1 VER_PCS Main

As the executive process of the PCS application, the VER_PCS main initializes the application, controls the main simulation loop, and generates a GUI and on-screen displays with which to control the application.

5.5.1.1 PCS Initialization

The VER_PCS Main function provides primary control of the simulation and all Performer functionality. An important aspect of the VER_PCS main is the initialization of Performer and respective shared memory structures. The VER_PCS main defines and creates internal data structures and shared memory structures to manage data across multiple threads. The VER_PCS main also initializes Performer and all local and shared memory variables.

After Performer is initialized, the VER_PCS main creates and initializes all support class instances. Instances are created for the Common_Renderer, IO_Mouse, IO_Keyboard, Selection_Manager, and PCS_Medigram_Manager. These classes perform the rendering, IO management, geometry selection, and communication functions for the VER PCS main, respectively.

The VER_PCS main handles loading all of the patient avatars used for the simulation. Dressed and undressed versions of each avatar are loaded into a Performer sub-tree that permits the current avatar to be interactively selected. This implementation decision extends memory

requirements and start-up time for the PCS application, but provides better run-time performance because loading geometry databases on demand is not required.

Also controlled by the VER_PCS main is task synchronization with the MSS. The VER_PCS main initially decodes command-line parameters to control the initial mode of execution of the PCS application. The PCS operates on or off the network. The on-network mode causes the PCS to broadcast MediGrams when the simulation is activated by the user. Off-network execution causes the PCS application to operate without sending or receiving Medi-Grams.

5.5.1.2 PCS Simulation Control

PCS processing is controlled from a main simulation loop. The simulation loop in the VER_PCS main is almost completely focused on processing user interface events. as shown in Figure 5-32. The loop completes initialization, and waits for a start simulation event from the user. When received, the VER_PCS main loop processes inbound and outbound MediGrams, triggers the Performer CULL and DRAW threads, polls for input events from the mouse and keyboard, updates the Renderer and GUI settings, and changes the virtual patient if required.

The VER_PCS main simulation loop exits when a terminate simulation event is received from the user. After exiting the main simulation loop, the VER_PCS main issues calls to *pfuExitGUI*, *pfuExitInput*, and *pfuExitUtil* to gracefully terminate and remove Performer data pool files created during processing. In addition, the VER_PCS main calls all non-null destructors for subordinate classes, and finally invokes *pfExit* to terminate Performer processing.

5.5.1.3 PCS Interface

The PCS rendering channel is divided into a GUI and a display area. The display area of the interface shows the current patient avatar. Using the *libpfui* trackball *pfiXformers*, the avatar may be moved and visualized from any vantage point under constant lighting. In addition, the PCS employs the Selection_Manager to permit selection and manipulation of patient avatars. This provides a capability to highlight the geometry of the avatar, which provides information about its complexity and surface geometry details. This also provides an interface capability for improved interaction with the avatars.

```
Create virtual patient avatar sub-tree
Read Start Simulation event
PCS_Medigram_Manager sends initial PR MediGram
while (!done) loop
        If on network:
                If time interval since last DT MediGram multicast exceeds threshold:
                        PCS_Medigram_Manager sends PV/PR MediGrams
                        PCS_Medigram_Manager retrieves current DT MediGram
        Initiate Cull and Draw traversals
        Poll Mouse
        Poll Keyboard
        Update View in Common_Renderer
        Update GUI
end loop
```

Figure 5-32. VER_PCS main simulation loop.

A layout of the PCS GUI is provided in Figure 5-33. Because the PCS interface is not immersive, the GUI occupies 30 percent of the total PCS screen space. The intent is to provide two types of functionality: simulation control that affects how the PCS executes, and virtual patient control that permits patient physiological data to be viewed and modified. The simulation control widgets are used to terminate the application, hide the GUI, reset the visual display, display Performer statistics, display MediGram statistics, display simulation clock, and start and stop the VER simulation. The virtual patient controls include widgets to select a virtual patient, clothe and de-clothe the current patient avatar, and provide sliders to change and view the current vital signs of the virtual patient before and during the simulation.

Figure 5-33. PCS GUI implementation.

The entire interface of the PCS application is implemented in the VER_PCS main. This implementation decision is made because (1) consolidating the GUI with a central view of the application make programming the interface easier, because the GUI has visibility to all classes,

(2) the GUI must be updated by the DRAW thread and would require considerable effort to package in a separate control class, and (3) the geometry management of switching patient avatars is minimal, and does not require a separate class. The GUI panel contains additional space for adding future capabilities as they are identified.

5.5.2 Script Manager

The Script Manager automates the task of configuring the virtual patient before each simulation. Configuration data is written to an external file that is read-in to load initial patient information and data. The Script_Manager class provides the methods necessary to read the external file, parse the fields, and write the data to the CODB where it is used to create Patient_Record and Patient_Vitals MediGrams. Functionality is limited to accessing external data files and storing values in appropriate fields of the PCS CODB.

5.5.3 PCS_Medigram_Manager

The PCS_Medigram_Manager class, like the MSS_MediGram_Manager class, is implemented to encapsulate the functions of Sheasby's Medical Network Manager class. Using this encapsulation approach, the interface to the VER_PCS main is insulated from potential changes to the Medical Network Manager Interface. The interface provides methods that perform the following PCS-specific functions:

1. Read the local PCS CODB.
2. Package information into correctly formatted MediGrams.
3. Broadcast the MediGram package to the MSS MediGram Manager.
4. Receive inbound MediGrams from the MSS.
5. Store formatted MediGrams in the PCS CODB.

The PCS_Medigram_Manager creates and maintains three CODB pointers: one for assembling Doctor_Treatment MediGrams, one for accepting Patient_Vitals MediGrams, and one for accepting Patient_Record MediGrams. The PCS_Medigram_Manager includes methods to send the Doctor_Treatment MediGram, and check for the receipt of new Patient_Vitals and Patient_Record MediGrams using the CODB pointers.

The methods provided in the PCS_Medigram_Manager class are shown in Figure 5-34, and parallel those created for the MSS_Medigram_Manager class. However, the definition of each method is defined based on MediGram operations relative to the PCS. Thus, the Initial-

ize_Outbound methods clear the Patient_Record and Patient_Vitals MediGram areas in the local PCS CODB, and the Initialize_Inbound method clears the Doctor_Treatment area in the local PCS CODB. The PCS_Medigram_Manager also includes methods to enable and disable the PCS_Medigram_Manager, and to get and show all of the MediGram data in the CODB.

PCS_Medigram_Manager
PCS_Medigram_Manager *~PCS_Medigram_Manager* *Initialize_Inbound* *Initialize_Outbound* *Initialize_PatientRec* *Update_Outbound_Automatic* *Update_Outbound_Console* *Retrieve_Inbound* *Enable* *Disable* *Show_Patient_Vitals* *Show_Doctor_Treatment* *Show_Patient_Record* *Show_Last_PV_Sent* *Get_Patient_Vitals* *Get_Doctor_Treatment* *Get_Patient_InitRec* *Get_Last_PV_Sent*

Figure 5-34. PCS_Medigram_Manager class methods.

A unique capability of the PCS_Medigram_Manager is the ability to update the outbound Patient_Vitals MediGrams two different ways. The Update_Outbound_Console method updates the Patient_Vitals record from the GUI directly, thus permitting the evaluator/user to change the patient vitals data manually. The Update_Outbound_Automatic method automatically updates the Patient_Vitals data from a non-GUI source, such as a patient physiology model.

5.6 Geometry and Texture Maps

To implement an immersive training VE, the doctor trainee must be able to see and interact with a 3-D representation of the ER facility. As a result, 3-D geometry is required to generate a visible room and associated emergency medical apparatus. Texture maps are used to improve the realism of the ER scene. There are two important implementation details that impact building a synthetic ER: building geometry databases that define the ER and associated

contents, and procuring geometry databases to represent the virtual patient avatar. The implementation details associated with each are discussed in this section.

5.6.1.1 Apparatus Geometry Databases

Geometry databases for the VER facility and apparatus are created according to two processes. The "Initial Modeling" process is used to create complete, texture-mapped models from pictures, photographs, and other information sources. The "Articulated Features" process is then used to extract articulations from the complete models so that they may be rendered in the Performer environment as articulated features. Both processes are discussed in this section.

5.6.1.1.1 Initial Modeling Process

The Initial Modeling process provides a consistent geometry database construction template for constructing all VER geometry. This process, as depicted in Figure 5-35, extends beyond mere construction of wire-frame geometry. Rather, the goal is to develop a database that integrates wire-frame geometry with correct colors, materials, texture maps, and internal structure.

The first step in the Initial Modeling process is to obtain information about the objects to be modeled. All geometry databases created for the VER MSS are based on information and measurements taken from actual objects. This information is compiled from three sources:

1. Photographs and measurements recorded during the hospital site visits.
2. Medical apparatus brochures and advertisements from current medical publications.
3. Drawings and illustrations.

This visual information provides a basis for developing the MSS geometry databases, and is useful to obtain general proportions of objects, relative dimensions, and unique physical characteristics.

The next step is to create wire-frame geometry of the target object, using the measurements and photographs. This task uses the measurements from the preceding step to generate polygon groups that represent the exterior of the object. The model wire-frame is systematically developed using this data. Color is then applied to polygon groups, and materials are added to simulate the physical surface properties. While coloring polygons is a straightforward, application of materials is a process of trial and error. For example, materials may be used to control the diffuse, ambient, and specular lighting properties of selected geometry, and may also be used

to make geometry emissive or transparent (transparent geometry must be defined last in the DWB hierarchy). In DWB, materials are applied per face, and when smooth surfaces are required, material illumination is based on vertex-average lighting. This permits shading functions to be blended, resulting in a less faceted surface. Such techniques are particularly useful for rendering smooth surfaces, such as rubber and skin.

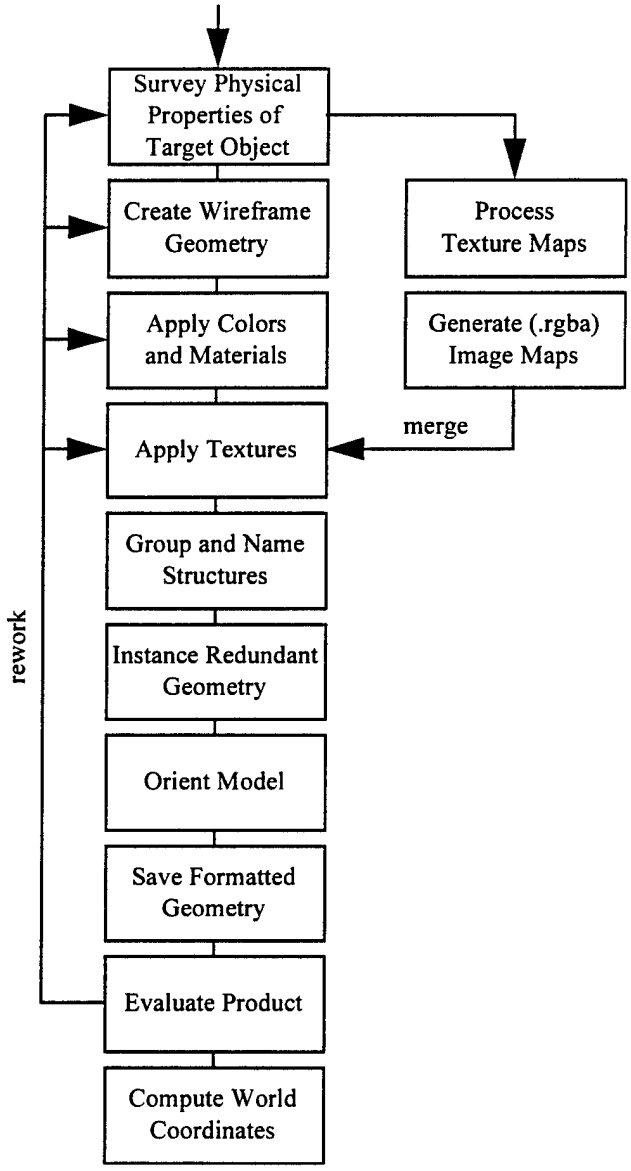


Figure 5-35. "Initial Modeling" process for creating VER geometry databases.

A parallel process to developing the wire-frame geometry is developing texture maps from the photographs or other image sources available. Images are scanned or imported into an image processing program, and retouched to improve image quality. The retouched image is then saved, transferred to the SGI environment, and converted to an (.rgb) or (.rgba) file. The resulting texture map file is imported into DWB, where it is applied to geometry as required. All texture maps that require use of the scanner are processed in the Macintosh environment using Adobe Photoshop, and converted using the from* utilities available in the SGI environment. Texture maps that are generated manually (not from a photographic source), such as lettering, are created using GIMP v1.0 in the SGI environment.

A variety of texture map techniques are possible. A simple approach that consistently yields acceptable results is applying detailed texture maps (such as mappings with words or lettering) onto separate polygons that are raised no more than 0.5 mm over the geometry to be texture mapped. Thus, detailed texture maps are applied as decals. Raised polygon surfaces prevent the "flimmering" anomalies associated with coplanar 3-D geometry [IRIS96]. If surrounding geometry must blend with decal mappings, a separate texture map with just the background color is also created and modulated onto the surrounding geometry. Texture maps that contain a repeating pattern are applied to geometry on a per-plane basis, to control the degree of pattern repetition.

Texture memory is a limited resource in the SGI environment. Performer 2.0 allocates texture memory in even powers of 2, so texture map files must be cropped to minimize wasted memory. A technique widely used in the VER geometry is combining different texture maps into a single texture file. For example, all of the lettering decals for an object are taken from different regions of the same texture file. This is accomplished by creating a master decal polygon and selectively cutting it to size to create the texture decal geometry. This technique dramatically reduces wasted texture memory, simplifies configuration management of texture map files, and minimizes Performer load times.

The settings used with GIMP to create textures with lettering follow a common format, which is effective in both DWB in Performer. The format shown in Table 5-1 shows the settings used to create the texture maps for the Defibrillator, and are representative of the settings used to create all other maps.

Defibrillator Texture Map Settings Using GIMP v1.0			
Text Detail	Value	Text Detail	Value
Font	"Helvetica 16-25"	Slant	"r"
Anti-alias	"10"	Width	"*"
foundry	"adobe"	Spacing	"*"
Weight	"bold and semi"	Plug-in Format	"SGI"

Table 5-1. Representative texture map settings in GIMP.

After texture maps are applied, the 3-D geometry is grouped into structures that are easier to manipulate. Structuring is not performed until texture maps are applied, because texture mapping often requires the construction of additional geometry. Creating structure groups also requires assigning names to the individual groups. The naming is an easy way to document the model, and because the names are often visible within the Performer sub-tree, appropriate names helps to understand and troubleshoot construction of Performer sub-trees.

After geometry is grouped and named, opportunities to instance redundant groups are usually apparent. Instancing minimizes geometry (and hence database size), and it also propagates the attributes of the reference object throughout the database automatically [VINC95]. Thus, instancing is a common feature of the VER geometry. An apparent instancing bug in the (.dwb) loader may be circumvented by saving the geometry in Z-up orientation (which is native to Performer) instead of the Y-up orientation (which is native to DWB). This orientation makes positioning the geometry easier, because the axes in DWB and Performer are correctly aligned. Another nuance in DWB is the need to define reference groups above instance groups in the DWB database hierarchy. Failure to do this causes errors with Performer (.dwb) database loader.

The next step in the development process is to orient the model on its own local coordinate origin. This is accomplished by translating the object center to the origin (using the DWB grid), and orienting it for proper alignment with a major axis. The local coordinate origin is thus the centroid of the model. This important step is required to ensure that Performer correctly orients the model; off-center models do not rotate correctly in Performer applications.

The final development step is saving the geometry database with the correct settings. The (.dwb) geometry databases are written with the current visual settings of DWB, so settings must be applied to the current DWB session to be correctly written to the geometry database file. Most VER geometry is saved with back-facing on, so that only front-facing polygons are visible. In addition, Z-buffering is always used, and all geometry shading is illuminated. These proper-

ties are important to achieve the proper visual appearance within Performer. Additional configuration requirements include orientating the model as Z-up (as previously described), and setting the units of measurement to meters. Finally, texture mapping must be enabled. The file save format is the common (.dwb) file format. These options are summarized in Table 5-2.

Attribute	Value
Z-Buffer	Enabled
Backface	Enabled
Shading	Illuminated
Orientation	Z-Up
Units	Meters
Textures	Enabled
Illumination	Enabled
Material Binding	Per-Face /Vertex-Averaged

Table 5-2. Geometry database save options in DWB.

After a geometry database is fully developed, it is evaluated for quality and appearance. This subjective step is performed using the IRIS Performer *perfly* application to visualize the geometry from various orientations and lighting conditions. The *perfly* application verifies that the geometry is readable by the Performer (.dwb) loader; ensures that all materials, colors, and texture maps are correctly applied; and is useful for visually checking that all geometry is correctly defined. If discrepancies are found with any aspect of the model, the process reverts to the appropriate development stage to resolve the problem. When the database is acceptable, it is released for use by the VER MSS application.

Construction of individual models alone does not satisfy the scene integration requirements of the VER MSS. The placement of each object in the final Performer scene requires information about where the objects are to be located, how they are to be scaled, and how they are to be oriented. Unfortunately, Performer does not provide a capability to interactively define the structure of graphical scenes. The approach adopted for the VER is to generate a scene layout model using DWB. This model file integrates several individual models into a single file, so that relative placements may be defined. By orienting the layout about a common origin, coordinates of individual models are extracted for use as world coordinates in Performer. These coordinates are recorded in a central header file that defines properties for all MSS geometry.

5.6.1.1.2 Articulated Features Process

After the world coordinates of the geometry database are known, the geometry database is immediately integrated into the scene using the Model class. For static geometry with no articulations or moving parts, the development work is complete. However, many objects contain elements that require motion. For example, buttons must be moved relative to the object to which they belong. In these cases, the movable geometry must be separated into distinct databases that may each be independently moved. Thus, the work performed to create a complete geometry database must, in a sense, be "undone" so that all moveable objects are unique geometry databases. The "Articulated Features" process shown in Figure 5-36 is used to consistently process geometry databases that contain movable parts.

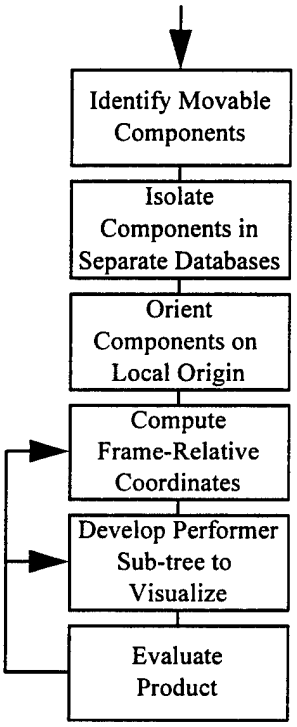


Figure 5-36. "Articulated Features" process for partitioning VER geometry databases.

The process for partitioning a model revolves around the idea of maintaining the modeling coordinate origin of the complete object. The complete object is decomposed into a frame (the main housing) and the articulated feature components (such as a button, or a wheel). Each articulated component is saved to a separate file and centered on its own coordinate origin. The

frame object is saved without any of the component objects, but is not re-centered on its local origin.

Using the complete model as a guide, the component objects are imported into the file containing the frame object. The component objects appear at the origin (usually somewhere inside the frame object). The last step is to compute the translation and orientation necessary to align the component object to its previous position, relative to the frame. This is done using the same process for computing world space coordinates. These "frame relative" coordinates are recorded for each component, and recorded for use by Performer.

When all component objects are isolated, and all frame-relative coordinates are computed, the geometry will no longer be usable in the Performer environment. An apparatus object class must be written to construct the Performer Sub-tree required to view the integrated object. Because the coordinates of the component objects are relative to the frame object, they should be child nodes of the frame object so that proper spatial relationships are maintained.

The collection of dynamic geometry databases with articulated features is presented in Appendix A. Static geometry databases developed for the VER are presented in Appendix B.

5.6.1.2 Patient Avatar Geometry Databases

In addition to the equipment, the patient avatar represents another modeling challenge. To develop a real-time simulator, the tradeoff between rendering speed and geometric fidelity must be negotiated. Empirical tests show that avatar models with greater that 18,000 polygons (total) degrade performance to the extent that movement and other functions are no longer easily accomplished.

Avatars from a variety of sources were investigated for use within the VER applications. Initial attempts to locate medium-fidelity patient avatars at various internet sites were unsuccessful. Avatar geometry typically found at most geometry repositories was low-fidelity and difficult to edit. Another source for avatar geometry, the visible human data set, is too complex and only provides one patient choice. Thus, additional work to generate a visible human was terminated.

Patient avatars were ultimately purchased from the Zygote Media Group. The avatars are first generation wire-frame models generated from full 3-D body scans. Each model is retouched, simplified, and provided in the common (.obj) and (.dxf) formats. The model

inventory includes undressed avatars for a male of "ideal" proportions, a female of "ideal" proportions, an infant, an obese male, and a muscular male. In addition, the inventory includes fully dressed ideal male and female avatars, and two additional models representing lower fidelity, "stylized" male and female avatars. Each avatar is moderately complex, with polygon counts ranging from 5,000 to 14,000 polygons. Thus, this collection improves the capability for varied simulations by: (1) increasing the realism of the avatars; (2) adding the capability to triage and treat emergencies on both genders; (3) permitting simulations involving emergency pediatric care; and (4) facilitating acceptable rendering performance.

This solution also allowed research on other aspects of the VER to continue. However, additional work was required to configure the avatars in the collection for use in the VER applications. Because the patient models are not available in the (.dwb) format, a considerable amount of post processing was required to prepare and maintain the models World-space coordinates for all avatars are computed and stored in an independent avatar_defs header file, for use by the MSS and PCS applications. The collection of patient avatar geometry databases incorporated into the VER prototype is presented in Appendix C.

5.7 Conclusion

This chapter discusses the implementation of the MSS and PCS applications of the VER prototype, and the construction processes used to develop the synthetic ER geometry. The implementation is guided by the requirements and the design specified in Chapters 3 and 4. The focus of this effort with respect to MSS implementation is visual realism and intuitive user interaction. The focus with respect to PCS implementation is an intuitive and flexible application to control the virtual patient. In the next chapter, the results of this effort are presented with respect to the original design objectives.

6. Results

6.1 Overview

This chapter presents the results of the PCS and MSS system implementation. The capabilities of the MSS and PCS applications are described, followed by a summary of the performance characteristics of each application. Finally, the system requirements specified in Chapter 3 are revisited to verify the functional capabilities of the VER prototype.

6.2 MSS Capabilities

The MSS, when activated, starts the simulation with the trainee in the ER away from the apparatus. The trainee immediately has a wide-area view of the synthetic ER, as shown in Figure 6-1. The view consists of one or more of the dynamic apparatus scene elements described in Chapter 4. In addition, static geometry such as walls, floor, ceiling, and cabinets are included to complete the ER scene. The patient avatar is dynamically loaded, based on the avatar selected from the PCS application.

As shown in Figure 6-1, the geometry is arranged and properly scaled. Texture maps and materials are used extensively to provide a realistic environment. Lighting effects are limited to an infinite light source in the ceiling, and a directional local light source in each of the two overhead directional lights.

The trainee is free to move about in the ER. All movement is controlled by the mouse. Forward movement in the direction of view is accelerated with the left mouse, and reverse acceleration is controlled with the right mouse button. The center mouse button serves two functions: stop movement, and activate the Selection Manager on the object designated by the mouse. If an object is selected, these mouse functions overlap with those defined for the Motion Manager. To minimize problems, the 'M' keyboard key is used to selectively pause view motion.

The minimal GUI interface provided for the trainee is shown in Figure 6-2. This interface bar permits performance information overlays to be displayed, such as graphics performance and current MediGram information. Another useful aspect of the interface is the ability to see and reset the object last selected by the Selection_Manager.

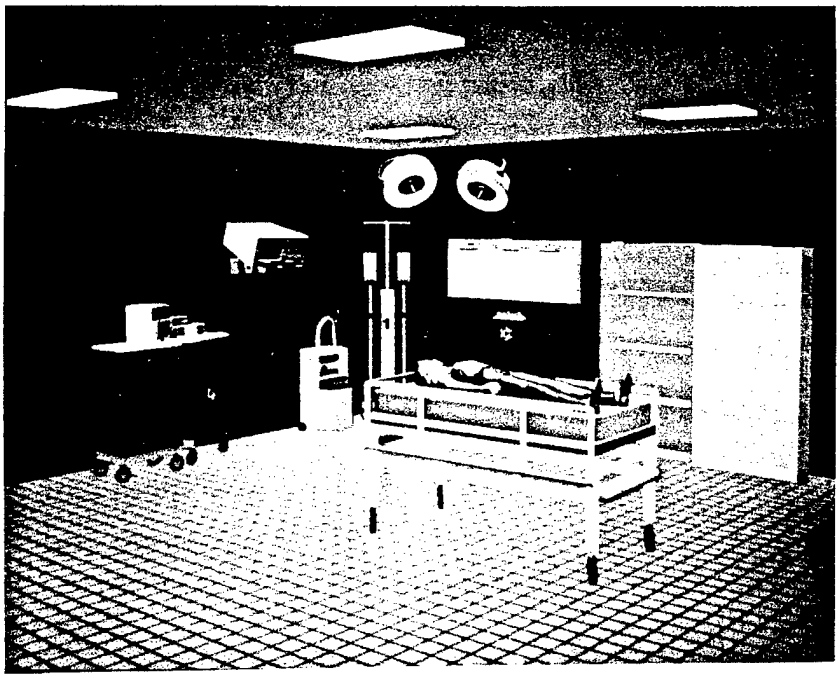


Figure 6-1. Wide-area view of synthetic ER.

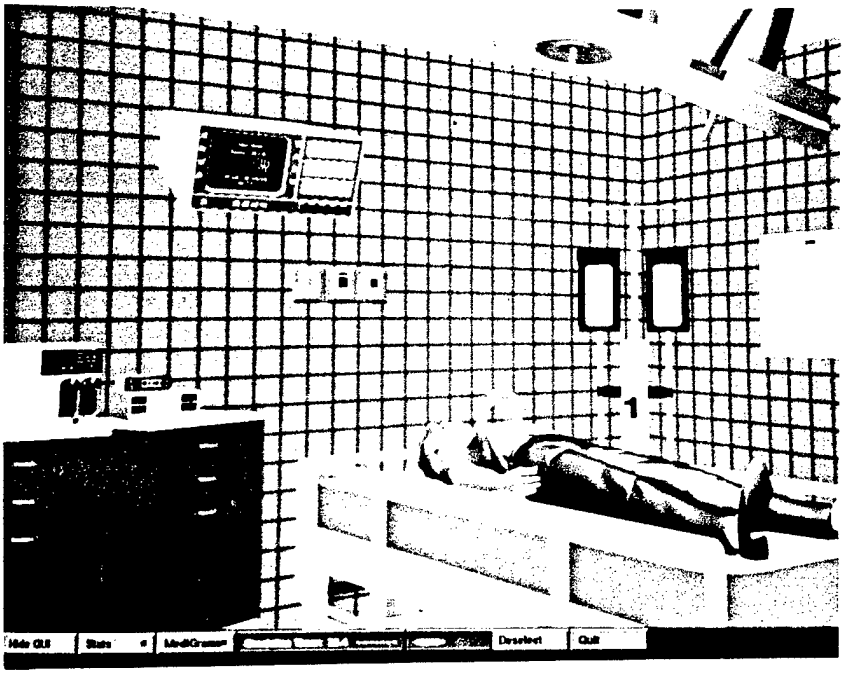


Figure 6-2. Patient evaluation, with GUI enabled.

These minimal interface capabilities are augmented only by the ability to hide the GUI and terminate the simulation. The GUI does not support triage, diagnosis, or treatment tasks. The objective is to deliberately minimize the interface.

A significant feature of the synthetic ER is the functional apparatus. As shown in Figure 6-2 and Figure 6-3, the directional lights may be used during the simulation to illuminate the patient and the x-ray backlights may be illuminated. The curtain may be drawn, and the cart drawers may be opened and closed. All functionality discussed in Chapters 4 and 5 is provided in the simulation. In addition, all of the digital monitors are functional, and may be used to assist with triage and diagnosis of simulated cases. Each monitor displays information true to its corresponding real-world counterpart.

Figure 6-3. Close proximity view of patient.

Considerable emphasis is placed on developing the geometry and apparatus classes for the digital components, so that it closely matches that used in actual ER settings. Some of the digital apparatus is shown in Figure 6-4. The displays of each device are updated using Medi-Grams received from the PCS. The control panels of the apparatus are functional and, at a minimum, permit each device to be turned-on and off. Connection wires and electrodes are omitted from the scene to minimize computational and rendering complexity.

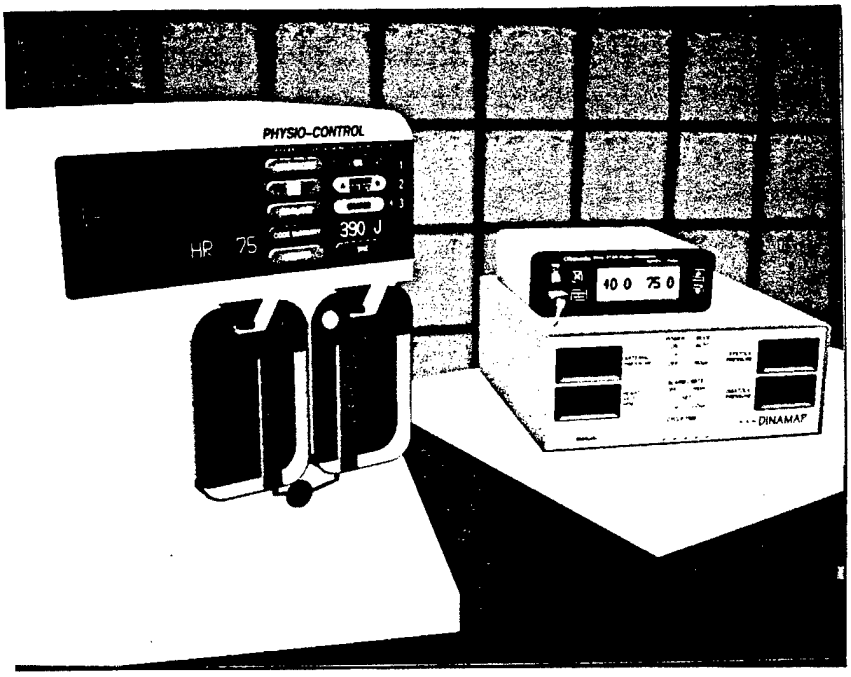


Figure 6-4. Digital apparatus updates via MediGrams.

Part of the capability of using the apparatus is the ability to move it into the proper position to render treatment. As shown in Figure 6-5, the Infusion Pump is relocated near the patient using the Motion_Manager. The control panel of the Infusion Pump is then accessible to the trainee. Motion is selectively enabled for the apparatus. Static geometry is not moveable, small apparatus may be lifted and oriented in all but the roll (Y) axis, and larger apparatus may be moved laterally and rotated about the heading (Z) axis.

Gravitational effects and inter-object collision are not implemented. Inertia is simulated using a "rests-on" dependency that causes moving objects to "carry" other objects resting upon them. Thus, in the case of the equipment in Figure 6-4, moving the right Crash Cart will "carry" the Dinamap, which in turn carries the Pulse Oximeter. These objects may be freely moved, but are linked hierarchically linked until a gravitational model is implemented.

Also missing are wires, tubes, and support mechanisms for moveable objects, such as the directional lights and the main patient monitor. The support geometry for these objects is omitted due to the limited implementation schedule of this project, the computational expense of rendering them in real time, and the small marginal improvement in realism that they would provide to the simulation.

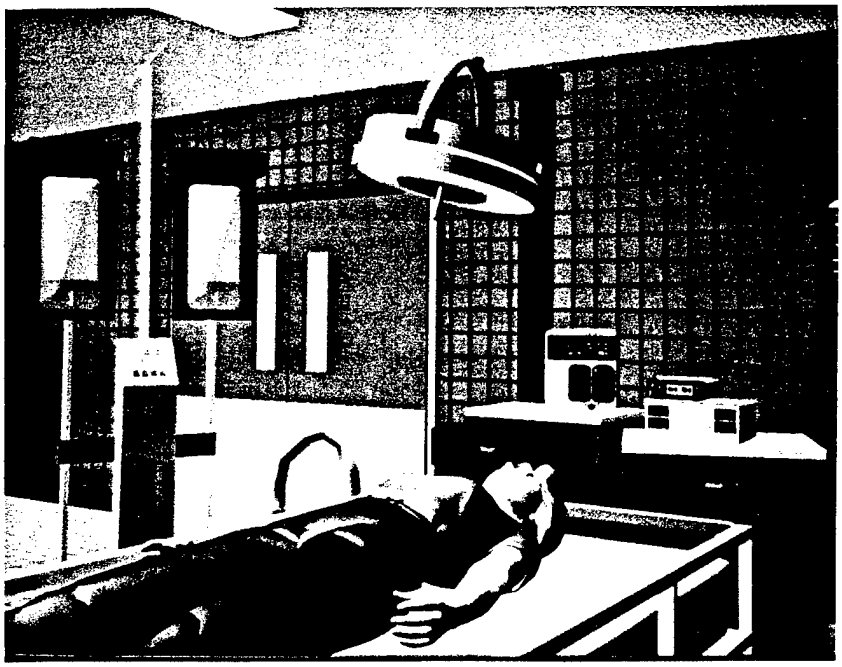


Figure 6-5. Apparatus relocated to administer treatments.

Movement of objects requires that they be selected using the Selection_Manager. Selected objects are circumscribed with highlight lines about their constituent bounding boxes. This highlighting provides a means by which the current object is visually determined. The Infusion Pump in Figure 6-6 is selected as an example.

The most important aspect of the synthetic ER is the ability to render treatments. Treatments are administered by interacting with apparatus in the ER. Each apparatus capable of producing a treatment contains a control panel. Most of these apparatus, such as the Defibrillator and Infusion Pump have control panels built into the object. The control panels are used to configure and activate treatments. Treatments are consolidated into an outbound Doctor Treatment MediGram, and then relayed to the PCS to update the virtual patient.

Not all apparatus have control panels. Control of non-electronic elements requires a pop-up window that permits the object to be configured. As shown in Figure 6-6, the IV Bag for the Infusion Pump is configured using a pop-up configuration window. With exception of their non-immersive nature, these windows provide a capability to rapidly expand the treatment capability of the MSS by abstracting away the details of manually preparing medical objects like IV Bags.

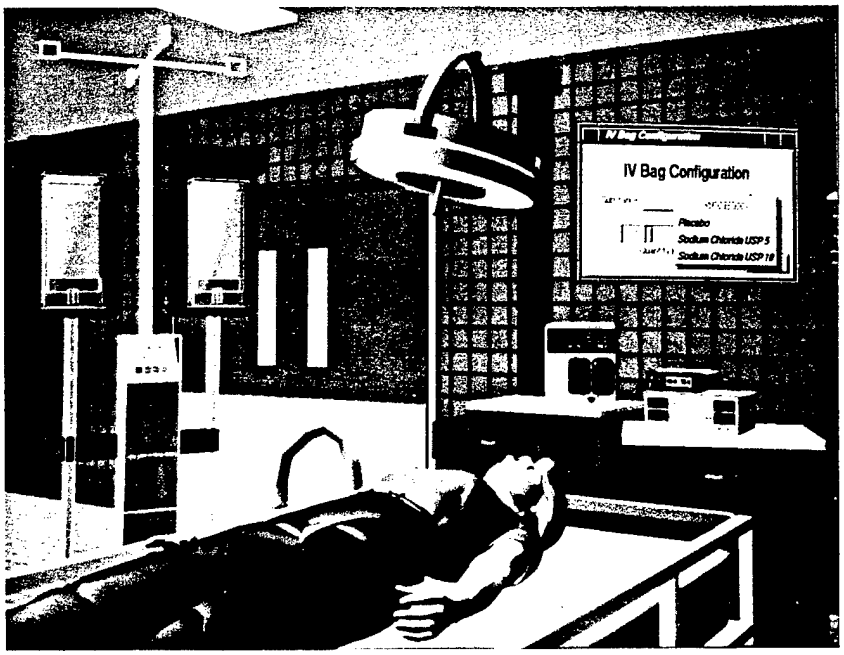


Figure 6-6. Selected apparatus and associated XForm control window.

The MSS is capable of providing infusion, defibrillation, and warming treatments to the virtual patient. These treatments are representative capabilities that demonstrate the feasibility generating Doctor_Treatment MediGrams from the MSS. As discussed in Chapter 7, support for additional treatments may be added by expanding the MediGram repertoire to communicate the necessary details.

6.3 PCS Capabilities

As presented in Chapter 5, the PCS is a graphical application. The interface is partitioned into two areas: the top 70 percent shows a current patient avatar in a Performer channel, and the lower 30 percent provides a control GUI in a separate channel below the avatar display. Figure 6-7 shows the basic PCS interface, with the "Ideal Woman" patient avatar selected. This view is typical of the PCS interface.

Above the GUI and superimposed on the overlay plane is the optional MediGram display HUD. This display shows the current Patient_Vitals MediGram to be broadcast by the PCS_Medigram_Manager, and the most recent Doctor_Treatment MediGram received by the PCS_Medigram_Manager. In addition, dynamic update flags are displayed to verify network activity during simulations.

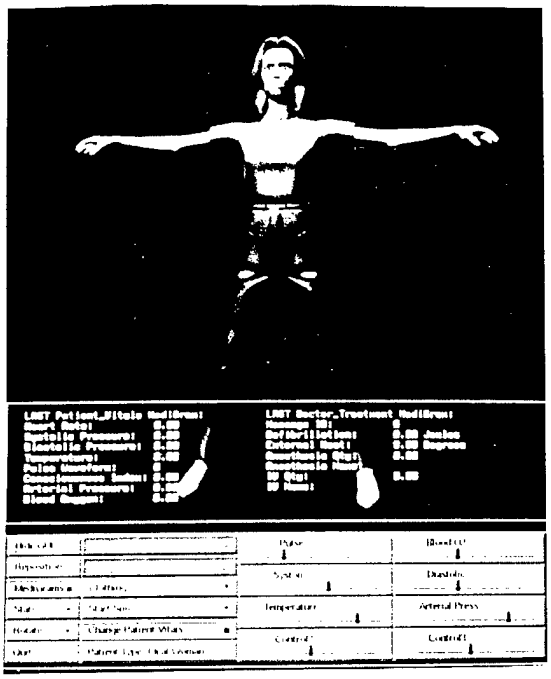


Figure 6-7. PCS Interface with MediGram view enabled.

As shown, the GUI itself is divided into two regions. The left side of the GUI is intended to control the operation of the PCS, select patient avatar geometry, and manage the PCS interface. The right side of the GUI is intended to monitor and control the vital signs of the virtual patient. Another useful viewing capability of the PCS is the display of the surface geometry of the patient avatar. Figure 6-8 illustrates the effects of highlighting surface geometry. Avatars may be selected using the GUI, and highlighted or un-highlighted by clicking on the avatar with the mouse. In addition, avatars may be moved to inspect specific features by moving the mouse. A close view of a different patient avatar is shown in Figure 6-9.

The narrow window in which the PCS executes may be expanded to include additional controls for the interface or the virtual patient. Ideas for expansion of the PCS are discussed in Chapter 6.

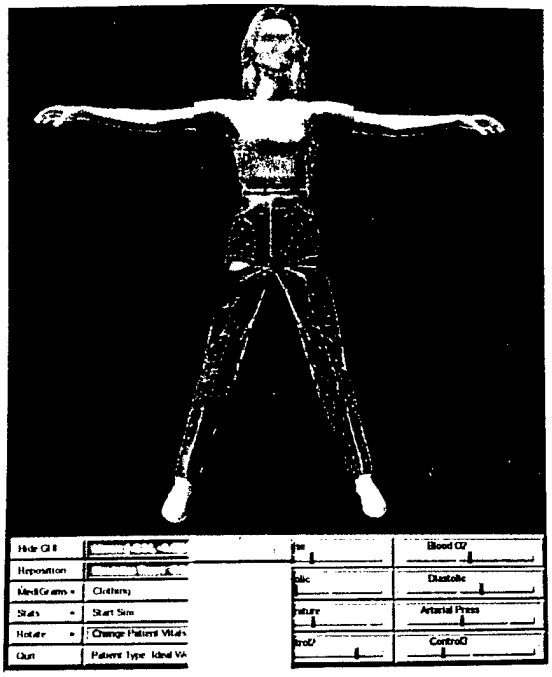


Figure 6-8. Avatar selection with geometry highlighting enabled.

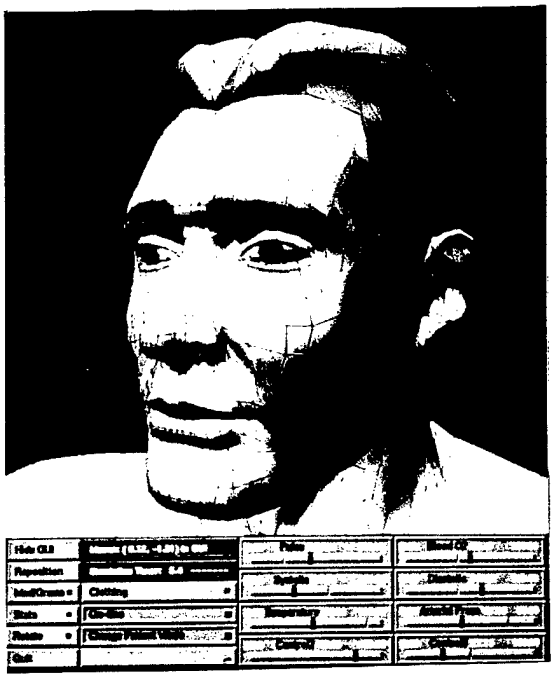


Figure 6-9. Selective avatar visibility in PCS interface.

6.4 MediGram Transfer

The basic VER operation requires constant MediGram traffic. The processing steps are fundamental to the functionality of the VER, and are summarized in Table 6-1. Initialization begins with the PCS, which broadcasts a Patient_Record MediGram containing initial simulation parameters. The MSS uses the initial MediGram to initialize the synthetic ER and trainee's interface. Doctor_Treatment MediGrams are then transmitted regularly to keep the PCS current. Similarly, Patient_Vitals MediGrams are regularly broadcast from the PCS to keep the MSS current. On a periodic basis, a Patient_Record MediGrams is also re-broadcast by the PCS to keep the MSS synchronized, and to permit the MSS to re-join ongoing simulations. Figure 6-10 depicts the MediGram exchange between PCS and MSS processes.

Time	PCS	MSS
0	PCS is activated on Host A	MSS is activated on Host B
1	virtual patient is selected, and initial simulation settings are specified from PCS interface.	MSS initializes
2	Start Simulation event is generated from PCS. Patient_Record and initial Patient_Vitals MediGrams created from configuration settings	MSS waiting for MediGram from PCS.
3	PCS activates virtual patient, updates interface with run-time display	MSS receives Patient_Record and Patient_Vitals MediGrams. Finishes initialization using data contained in MediGrams.
4	PCS checks for inbound Doctor_Treatment MediGrams every *x* seconds.	MSS graphics configured. Simulation active!
5	PCS updates and sends outbound Patient_Vitals MediGrams every *x* seconds	MSS checks for inbound Patient Vitals and Patient_Record MediGrams every *w* seconds
6	PCS updates and sends outbound Patient_Record MediGrams every *y* seconds	MSS updates and sends outbound Doctor_Treatment MediGrams every *w* seconds.
N	<process runs until termination event received>	<process runs until termination event received>

Table 6-1. VER MediGram exchange.

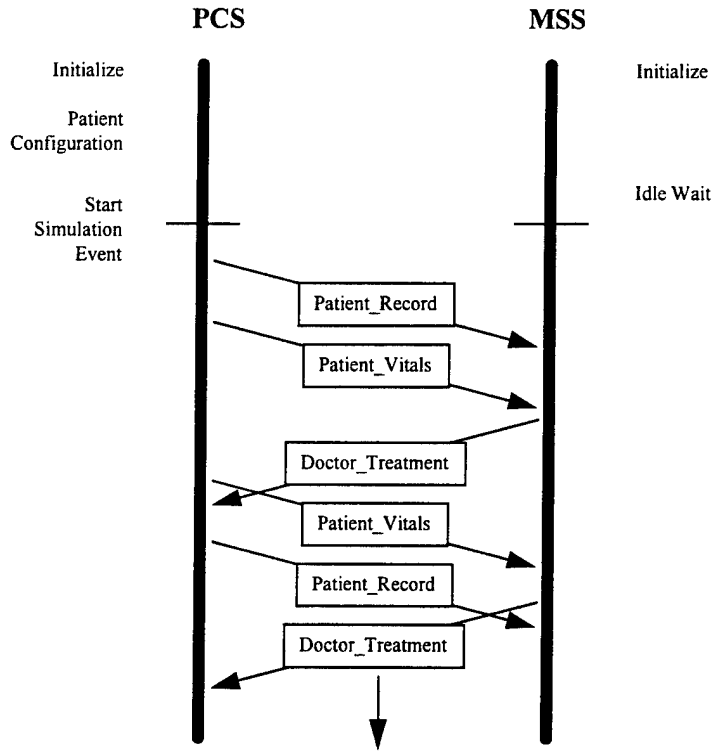


Figure 6-10. MediGram transfer during VER initialization and processing.

6.5 Performance

Performance of the VER applications is an important consideration, because the responsiveness of the applications is directly affected by the power of the underlying hardware. The VER applications are tested on three workstations available in the AFIT Graphics Lab. These include:

- Silicon Graphics 4 x 100 MHz R4400 CPU Onyx with Reality Engine 2 Graphics
- Silicon Graphics 2 x 100 MHz R4400 CPU Onyx with Reality Engine 2 Graphics
- Silicon Graphics 1 x 250 MHz R4400 CPU with High Impact Graphics

6.5.1 MSS Performance

The MSS application requires a substantial graphics rendering capability, due to two characteristics of the synthetic ER:

1. The high geometric complexity of the patient avatar, which adds approximately 12,000 polygons (consisting of one or more triangles) to the ER scene.

2. The dynamic elements of the scene (such as the apparatus) that require updates during each rendering frame.

These factors limit the hardware platform upon which the MSS should be run to predominantly high-end workstations. Although the MSS may be executed on any lab host with IRIX 6.2 and Performer 2.0, high performance workstations are best suited to the MSS application because it is compute-bound. Figure 6-11 shows the start-up times of the MSS on various hardware platforms available in the AFIT Graphics Lab. Figure 6-12 shows the maximum sustained frame rate achieved by executing the VER MSS on each hardware platform type. Figure 6-13 shows the maximum sustained DRAW thread times of the MSS on each hardware platform type. Based on this analysis and empirical tests, the 4 CPU Onyx workstation is the best choice for executing the MSS application. This platform is much more efficient at rendering the synthetic ER, as seen by the relatively high frame rate and the relatively low DRAW thread times. The small additional time interval required by the 4 CPU Onyx to load the MSS is not a serious performance concern.

6.5.2 PCS Performance

The PCS application requires less computation than the MSS, because there is no need to render and maintain an immersive environment. However, the PCS requires sufficient memory, to store all avatars in the VER inventory. Thus, the PCS is primarily a memory-bound application. An important performance criterion for the PCS is a reasonable frame rate and fast loading time of the avatar inventory. Like the MSS, the PCS may be executed on any lab host with IRIS 6.2 and Performer 2.0. Figure 6-11, Figure 6-12, and Figure 6-13 show the start-up, sustained frame rate, and DRAW thread performance of the PCS on the hardware platforms in the AFIT Graphics Lab, respectively. The PCS is less graphically intensive, in that steady frame-rate performance is not as important as in the MSS. Given the availability of these resources, the High Impact platform provides satisfactory rendering performance, and is the fastest at loading the inventory. The 2-CPU Onyx provides better rendering performance, making the interface more responsive, but requires more time to load the avatar inventory. Either host is sufficient for the PCS.

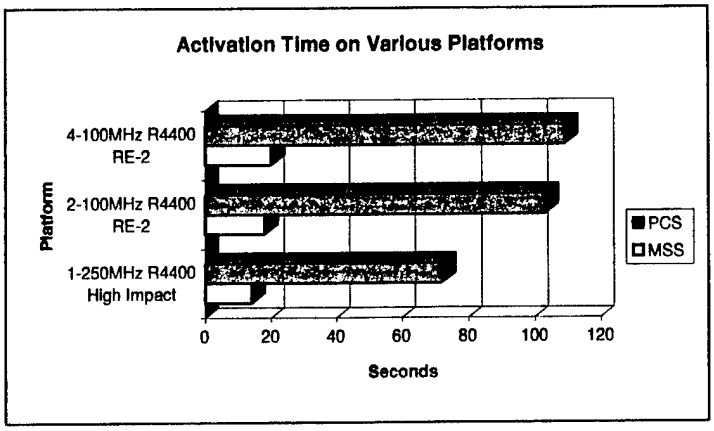


Figure 6-11. VER start-up times on various host platforms (shorter is better).

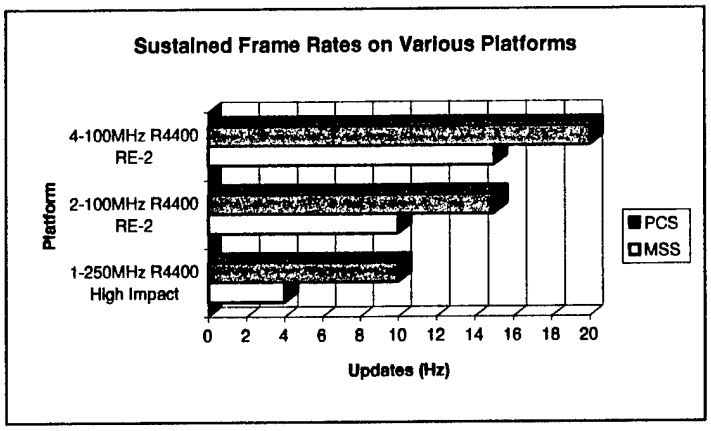


Figure 6-12. VER sustained frame rates on various host platforms (longer is better).

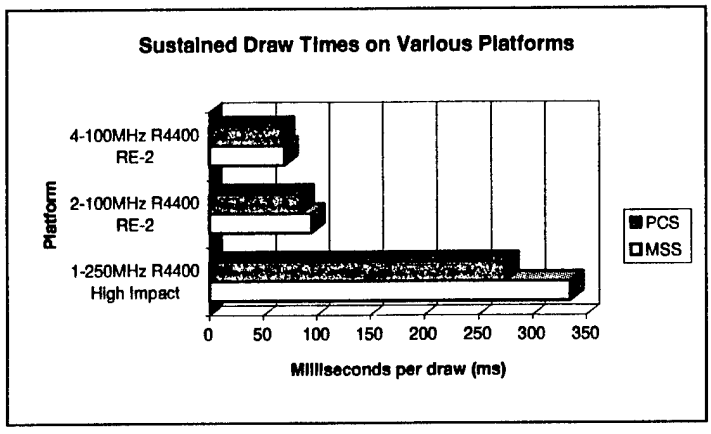


Figure 6-13. VER sustained draw times on various host platforms (shorter is better).

6.6 Requirements Traceability

To verify that the VER prototype implements the functionality specified in the initial requirements, this section re-visits the requirements and describes how they are satisfied. The original requirements outlined in Chapter 3 are summarized in Table 6-2, and annotated with how each is satisfied in the prototype.

6.6.1 DVE Configuration Requirements

The DVE requirements, established to ensure an extensible multi-entity simulation, were among the easiest requirements to address. The CODB is used for the DVE support architecture of this prototype. The participants of the simulation are the Patient Control Station (PCS) and the Medical Staff Station (MSS). Communication between these entities is supported by the exchange of MediGram messages. The MediGrams are queued, sent, and received by the specialized MediGram_Managers developed for each application type.

6.6.2 Virtual Patient Process Requirements

The Virtual Patient Process is implemented as the Patient Control Station (PCS). Most of the requirements are satisfied by integrating the appropriate controls into the PCS interface. Specifically, this includes the capability to monitor external patient vital signs, monitor simulation performance, select the virtual patient avatar, and control general simulation parameters. In addition, the PCS satisfies the requirement for accepting treatments using the PCS_Medigram_Manager to receive Doctor_Treatment MediGrams. Simulation scripts may be loaded at startup to initialize the patient settings. The PCS does not implement a patient physiological model, but provides a concise interface for integrating such a model when one is available.

6.6.3 Doctor Station Requirements

The Medical Staff Station is implemented to satisfy the requirements of the Doctor Station. The synthetic ER generated by the MSS is immersive, and permits the participant to view, move, and operate scene elements. The environment provides a representative apparatus capability for real-time monitoring and treatment of the virtual patient. Treatments are non-invasive. Implementation of interaction support tools, such as the Selection_Manager and Motion_Manager permit the trainee to interact with the scene elements as part of the training.

ID	Requirement Synopsis	Solution
	DVE Configuration	
1.1	DVE architecture	Employ CODB DVE Architecture
1.2	Virtual patient process	Patient Control Station (PCS)
1.3	Doctor station process	Medical Staff Station (MSS)
1.4	Medical DVE protocol	Designed MediGram formats based on DIS
1.5	Message Strategy	MediGram Managers at PCS and MSS
	Virtual Patient Process	
2.1	Physiological updates	Interface to physiological model provided
2.2	Accept treatments	PCS_MediGram_Manager polling
2.3	External patient vitals monitoring	Controls to monitor, change patient vitals
2.4	Simulation performance monitoring	Controls for performance, MediGram data
2.5	Scenario scripts	Script_Manager in PCS reads external files
2.6	Control simulation parameters	Controls to configure, start, stop simulations
2.7	Select patient avatar	Control to select and view patient avatars
	Doctor Station	
3.1	Trainee immersed	Synthetic ER created by MSS, ER_Manager
3.2	Identify and select scene elements	Selection_Manager permits on-screen picking
3.3	Move objects in all ranges of motion	Motion_Manager, motion up to 5 DOF
3.4	Functional ER apparatus	Apparatus classes in MSS, ER_Manager
3.5	Real-time, non-invasive treatments	MediGrams generated by Apparatus classes
3.6	Real-time patient monitoring	PV, PR MediGrams used by Apparatus classes to update apparatus in synthetic ER
3.7	Trainee in-room and following floor	Collision detection and ground following
	3-D Geometry	
4.1	Realistic Level I/II ER	Layout, equipment based on actual facilities
4.2	First-generation patient avatars	Implemented in (.dwb) format
4.3	Geometry for ER and apparatus	Created using process discussed in Chapter 4
4.4	Positioning and sizes accurate	Position, sizes based on actual facilities
	Support	
5.1	Support (.dwb) format	Support implemented, (.dwb) is sole format
5.2	30K polygons at 10 Hz, target 20 Hz	Sustained performance is 22K at 15 Hz
5.3	Support for Mouse, Keyboard, and Tracker input devices	All devices supported in MSS, Mouse and Keyboard supported in PCS
5.4	Uses existing Lab resources	Lab resources used for VER prototype

Table 6-2. VER requirements traceability table.

6.6.4 Geometry Requirements

Development of geometry for the VER represents a significant portion of the implementation effort. The emphasis is to provide a realistic and consistent geometry inventory for both

apparatus and patient avatars. All geometry databases include colors and materials, and most contain texture maps. Geometry is implemented to-scale and is properly oriented in the synthetic ER. The processes developed to create the basic geometry as well as that of dynamic apparatus includes steps to meet the geometry requirements. The patient avatar inventory consists of first generation avatars converted into the (.dwb) format.

6.6.5 Support Requirements

The VER prototype is implemented and tested using existing AFIT Lab resources, which includes Silicon Graphics workstations, Coryphaeus' Designer's WorkBench modeling software, and other ancillary software tools. Thus, the VER prototype supports (.dwb) geometry databases. Interaction with the VER applications is supported by the workstation keyboard and mouse. The MSS also support input from Polhemus position tracking equipment. As discussed in the previous section, the performance requirements of the VER applications are satisfied by the processing capabilities of existing Lab hardware.

6.7 Conclusion

As described in this chapter, the VER prototype provides a successful implementation of the original VER design. The MSS provides a synthetic Level I ER, complete with functional 3-D medical apparatus models. An emergency medical trainee operating from a dedicated Medical Staff Station (MSS) process participates in VER simulations with a remotely managed virtual patient. The trainee may interact with the VER, and all ER apparatus is functional in real time. Specifically, the trainee may view the vital signs of a virtual patient using the vital signs monitors, and may render rudimentary treatments to affect the medical status of the patient.

The virtual patient, which is controlled and monitored from an external Patient Control Station (PCS) console, may be initialized and continuously monitored as another participant. Patient vital signs and other physiological attributes may also be adjusted from the PCS console at any time. This makes the PCS console an ideal platform for evaluating the effectiveness of trainees in arbitrarily configured scenarios.

The results of this effort meet the original design requirements. While the capabilities of the design are far from complete, they demonstrate an initial capability that may be expended to provide a more mature training capability. Recommendations for improvement, and a discussion of the conclusions drawn from this work are discussed in Chapter 7.

7. Conclusions and Recommendations

7.1 Introduction

In this chapter, the primary accomplishments of the VER design and prototype are described. In addition, a description of the VER construction process is provided. Next, a summary of the success of the VER project, as measured against the thesis statement and the requirements enumerated in Chapter 3, is presented. Recommendations for future effort and research are then discussed, followed by a discussion of the relevance of the VER project to the Air Force and DoD mission.

7.2 Accomplishments

This research is the first effort of a several year project. As such, the starting point was entirely at the conceptual stage. The results of this phase include many tangible accomplishments that provide a solid foundation upon which future research phases can build. The significant accomplishments of this thesis effort specifically include the following:

- Designed the initial VER architecture, which include a virtual patient and a doctor station. This design is based on technical descriptions presented by Dr. Dumay and Dr. Godsell-Stytz in separate papers [DUMA96; GODS95].
- Researched and designed MediGram formats necessary to communicate emergency medical events within the VER DVE architecture.
- Implemented the VER MSS design to satisfy the doctor station requirements.
- Implemented the VER PCS design to satisfy the virtual patient process requirements.
- Designed and Implemented user interaction support objects for the VER applications, to include the Selection_Manager, Motion_Manager, Collision_Manager, Model class, and user interface designs.
- Developed a 3-D geometry database inventory for the immersive MSS ER, and an "Initial Modeling" process to consistently create additional VER geometry databases.
- Developed an "Articulated Features" process for isolating dynamic features of models for use in the Performer environment.

7.3 VER Construction Process

The steps taken to create the VER prototype may be summarized in eight general phases. These phases, shown in Figure 7-1, may be used to extend the functionality of the VER prototype.

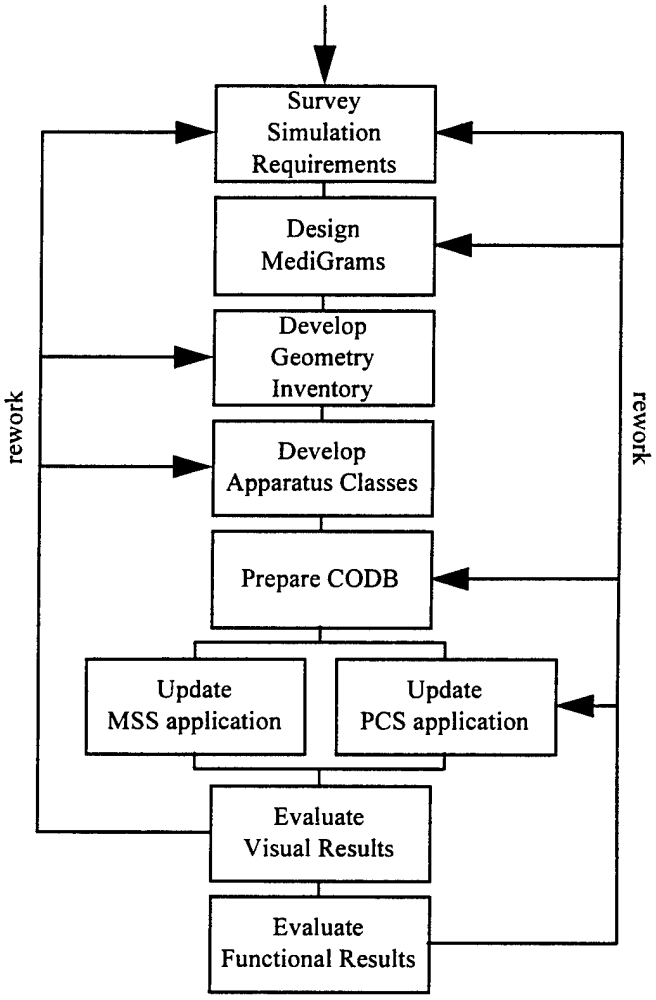


Figure 7-1. VER development stages.

The first phase was to survey the simulation requirements. This specifically addressed documenting the objectives and goals of the simulation. Preliminary design analysis was also useful at this early stage to determine which objectives were feasible. Requirements were stated and system-level design decisions were made.

In the second phase, the MediGram formats used to communicate between VER participants were designed. Early emphasis on the communication aspects of the training simulation helped identify how to develop the applications and geometry databases.

In the third phase, the geometry database inventory was created. Supplemental information was required in this stage, because modeling required measurements and photographs from the medical facilities. The development work was accomplished with the Initial Modeling and Articulated Features processes described in Chapter 5.

In the fourth phase, apparatus classes were developed to support the articulated features and additional processing requirements of the dynamic geometry databases. The apparatus classes make extensive use of Performer library calls. All apparatus functionality was programmed as apparatus classes for the corresponding object type.

The fifth phase prepared the Common Object Database to support all apparatus classes required for the simulation. CODB configuration involved creating unique CODB storage identifiers for all objects affecting the simulation. In addition, CODB areas were created to properly manage each MediGram type.

In the sixth phase, the VER MSS and PCS applications were updated to incorporated the static and dynamic geometry and apparatus classes. This involved integrating the classes and troubleshooting MediGram exchanges between the applications during testing. This stage also included integrating user interface objects with new functional capabilities provided by the new apparatus classes and MediGrams.

In the seventh phase, changes to the VER applications were evaluated for visual consistency and correctness. All testing was accomplished in the Performer environment using the corresponding apparatus classes and the *perfly* support application. Visual problems were corrected at the appropriate phase in the development process, and re-evaluated until corrected.

The eighth and final phase evaluated the VER prototype for functional correctness. For the MSS, the functionality is controlled by the apparatus classes and the ER_Manager. For the PCS, the functionality is controlled by the PCS_Main. Functionality was verified by creating and sending MediGrams between applications. Discrepancies were identified, corrected at the appropriate development stage, and re-evaluated until corrected.

7.4 Thesis statement revisited

The original thesis statement, or goal, can be subdivided into two objectives. The VER may be evaluated against these objectives to determine the overall success of the project.

The first objective was to develop an architectural design that permits simulation of emergency medical triage and treatment within a distributed virtual environment. This design, presented in Chapter 4, specifies a distributed medical simulation environment that incorporates most of the fundamental ideas originally discussed by Dumay and Godsell-Stytz [DUMA96; GODS95]. The primary results of the design was the specification of a virtual patient monitoring and control process (PCS), a separate and independent doctor trainee process (MSS), and a communication protocol that permits the processes to interact (MediGram protocol).

The second objective was to implement a functional prototype that demonstrates the feasibility of the designed architecture. The VER prototype was implemented according to the design presented in Chapter 4. The prototype provides a synthetic ER environment at the MSS, and a non-immersive control station for the virtual patient at the PCS. All communication between applications are performed using MediGram formats specified in the design.

During simulations, a doctor trainee at the MSS may interact with the synthetic ER and its contents. This specifically includes the ability to monitor patient vital signs and render non-invasive treatments for the virtual patient. An evaluator at the PCS may configure, monitor and change the physiological status of the virtual patient. These capabilities demonstrate the ability to communicate medical events within the VER prototype. Using dead reckoning and periodic transmission of MediGrams, the VER prototype proves that emergency medical simulation using DVE technology is possible and a feasible prospect for further research.

7.5 Recommendations for future work

While the features implemented in this version of the VER fulfill the established requirements, many potentially useful features are not implemented due to time constraints, resource limitations, or immaturity of existing technology. Where possible, interfaces for future work are annotated in the source code of the VER applications. Suggestions for additional work include (1) DVE Support enhancements, (2) improvements to the MSS human-computer interface, (3) improvements to the synthetic ER, (4) improvements to the PCS application, and (5) training and evaluation support. These categories are summarized in Table 7-1.

7.5.1 DVE Support

The distributed architecture of the VER prototype is sufficient to demonstrate the feasibility of the design. However, the functionality is limited to simulating the actions of one doctor trainee. The next logical step is to add the capability for complete medical staffs to collaboratively interact during training simulations. The PCS will not require additional modification to support multiple participants, but additional work will be required to design and implement this capability in the MSS. Each MSS must update doctor avatars and objects as they are changed by other participants. The processing necessary to support these features is in addition to that already required to drive the MSS. In addition, a capability for communication between MS Stations is needed. To relay communication messages, additional MediGram formats will be required to transmit messages. The MSS MediGram Manager must also be configured to accept both PV and appropriately addressed communication MediGrams from other MSS participants.

Functional Area	Proposed Work
DVE Support	Multiple MSS participants
	Expand MediGram repertoire to include more treatment types, inter-participant communication
	Federation Object Model (FOM) for medical HLA Federation
MSS Human-Computer Interface	Replace interactive XForms interface with immersive interface capability
	On-line help for simulation functions
	Force feedback interaction devices
	Audio cues
	Voice commands
	Hands-on interaction
MSS synthetic ER	Inter-object collision detection
	Environmental physical effects
	3rd Generation patient avatars and organ models
	Geometry databases and associated support classes for additional ER apparatus
	Doctor and medical staff avatars
PCS application	Patient physiological models
	Patient medical response scripts
	Capability for dynamic patient scarring
	Patient database capability

Table 7-1. Proposed VER modifications.

Figure 7-2 depicts the configuration of the proposed Phase 2 multi-participant VER, which incorporates a single Virtual Patient and 3 MSS participants. The figure shows the more complicated nature of communications in a next-generation VER, which includes the multicast patient information from the PCS to all MSS participants (white MediGrams), the need for inter-MSS communication messages (gray MediGrams), and potentially simultaneous Doctor Treatment MediGram updates to the Virtual Patient (black MediGrams). Beyond adding support for multiple participants, the existing MediGram repertoire should be expanded to include other MediGram formats that contain more patient status parameters and an increased variety of doctor treatments.

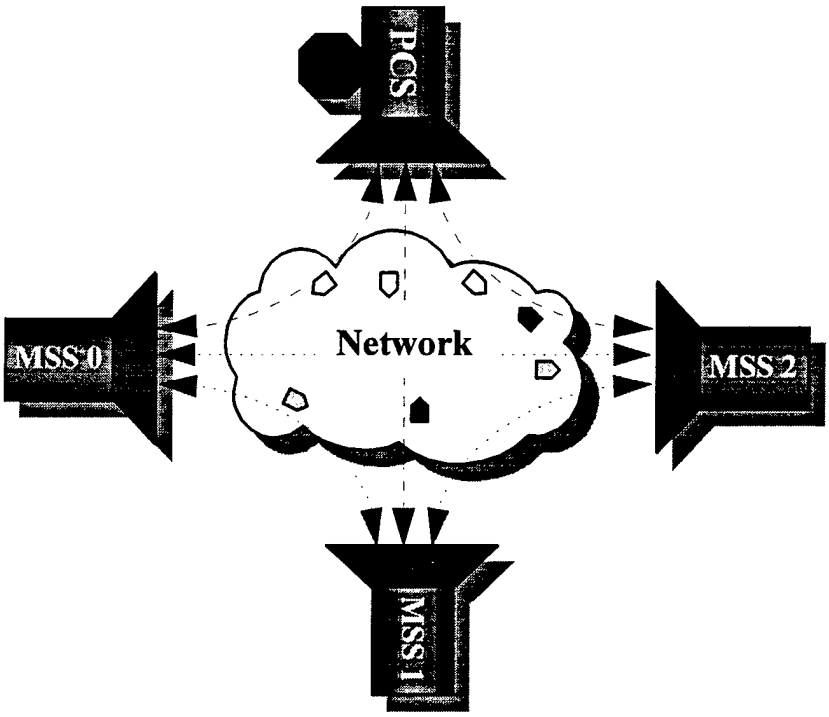


Figure 7-2. Proposed multiple MSS participant configuration.

A final improvement with respect to the DVE architecture is the migration to the HLA architecture. Using the existing MediGram formats, a federation for emergency medical simulation should be defined, and a corresponding Federation Object Model should be created. Compliance with HLA will permit the VER to evolve with changes in DoD modeling and simulation policies, and presents an opportunity for large-scale inter-operability with other HLA-based simulations.

7.5.2 MSS Human-Computer Interface

Additional improvements should be made to the human-computer interface of the MSS station. First, research should be conducted to find an immersive replacement for the XForms interface. This will simplify the interface by eliminating multiple pop-up form windows outside of the ER. Another improvement is integrating an on-line help capability into the on-screen portion of the MSS interface. This permits supplemental information about the interface to be displayed when needed.

The capability to permit lower level, "hands-on" interaction is needed. The ability to support this depends on identifying efficient collision detection algorithms, kinematics algorithms, and object selection techniques. Moreover, the inability to naturally interact with virtual objects has remained a long-standing problem, despite innovative work on the subject [MOCH92; HUAN95]. However, new paradigms such as the Virtual Workbench appear to be very promising in the medical VE domain [POST96a; POST96b].

The interface may also be improved by adding other sensory capabilities. Despite the lackluster capabilities offered by current technology, the state of emerging force feedback devices should be investigated. Devices such as the Phantom may be used to provide an initial haptic capability. Audio Cues are another interface capability that should be implemented into the VER MSS. This might include the capability to generate various sounds and noises such as dialog between trauma team members, apparatus, and patients. Finally, a voice command capability should be investigated. Voice commands may provide a natural interface that, if developed to the appropriate level of sophistication, can provide a good alternative to the non-immersive XForms interface previously discussed.

7.5.3 MSS Synthetic ER

Realism is a top priority in most medical VEs, because the more realistic the simulator, the better the training. As suggested by Hon, "if physicians and surgeons practice on many patients in a "realistic" simulator environment, they will definitely improve their performance on their first human patients [HON96]." Thus, many recommendations are intended to improve the level of realism of the synthetic ER.

For example, collision detection is required in the VER prototype. Similarly, other environmental effects such as gravity, inertia, and momentum must be integrated. Adding these

effects will increase the visual credibility of the ER, but so far have been computationally prohibitive. Another recommended improvement is to introduce 3rd generation patient avatars. As discussed in Chapter 2, third-generation patient avatars exhibit realistic appearance, correct inverse-kinematics, and tissue deformation properties. Such avatars will dramatically improve the usefulness of the VER, because the complex technical problems associated with hands-on medical treatment will not be exacerbated by unresilient first generation avatars.

Yet another suggested improvement is adding more ER geometry. Many, but not all, of the primary apparatus for emergency rooms are included in the VER prototype. Missing are the patient respirator, several in-hand tools (such as scalpel and scissors), and treatment related objects such as bandages. These models will become necessary as the VER matures. Complicated models, such as hoses, tubes, and wires must also be modeled when graphics rendering performance is improved.

Moreover, the synthetic ER layout should evolve as medical technology evolves. Contemporary ED facility configurations are difficult to find. A potentially useful information source is the *Journal of Emergency Nursing*. This professional bimonthly publication of the Emergency Nurses Association includes a regularly-run section that profiles the general layout and instrumentation of Emergency Department facilities from around the country. From the descriptions and many photographs provided, new directions in ED layouts and apparatus are easily discernible. This information, in turn, can be used to keep the VER facility layout and associated equipment models current.

Finally, an avatar for the doctor trainee should be implemented, so that a more believable first-person perspective is achieved. That is, the trainee should see parts of his or her virtual body during simulations. Later, as the capability of input devices improve, a mechanism may be introduced to permit individual movement of arms and hands as part of the interaction.

7.5.4 PCS Application

One of the most important recommendations for future work is integrating a patient physiological model into the PCS. As shown in Equation 7-1, the patient physiological model must permit the patient physiology to be updated based on current physiological state and medical treatments received.

$$\text{Patient\_Status}_{new} = \text{Physiological\_Model}(\text{Patient\_Status}_{previous}, \text{Treatments}) \qquad (7\text{-}1)$$

This model could be used to update the interface of the PCS as well as the patient information monitors in the MSS synthetic ER. A physiological model can be directly integrated into the PCS station with no additional innovation. It will be many years before such a model is mature enough to pass the "Turing Test," whereby the physiological model cannot be distinguished from an actual patient [HON96]. However, even a low fidelity initial capability should be implemented as soon as possible. An interim capability that is potentially useful is scripting patient responses to treatments received from the MSS. Response scripts provide a structured, but useful simulation capability, because the virtual patient can be updated in real time automatically.

Another recommended PCS improvement is to introduce a capability for adding surface details to simulate scars and wounds on the current patient avatar. This capability will be dramatically simplified if third generation patient avatars become available. The graphical PCS interface permits adding this capability without major redesign of the interface.

A final recommendation for improving the PCS is the addition of a patient database. A database would provide a way to store various patient records, which could be made available to trainees when needed for diagnosis and treatment. Additionally, medical history and other patient background information could also be included in the database. The database could be implemented at the PCS and queried from the MSS using appropriately formatted MediGrams.

7.6 Utility to the Air Force

This thesis has addressed many technical issues associated with the design and implementation of the VER prototype. An important question that must also be answered is "What impact does the VER have on the mission of the Air Force?" According to Dr. Satava:

> The military has a need to decrease combat casualties, and new advanced technologies are being utilized to reduce these casualties, including virtual environments as a planning and training tool. The saving of lives on the battlefield requires training in remote diagnostics, triage, casualty evacuation, first aid, patient stabilization, and specific combat trauma surgical procedures [SATA96b].

DARPA has sponsored the VER project to better realize these training and readiness goals. The Air Force is a beneficiary of the VER research, because the VER implements an initial capability that will ultimately provide faster and more effective trauma management skills training for mobile military field hospital teams.

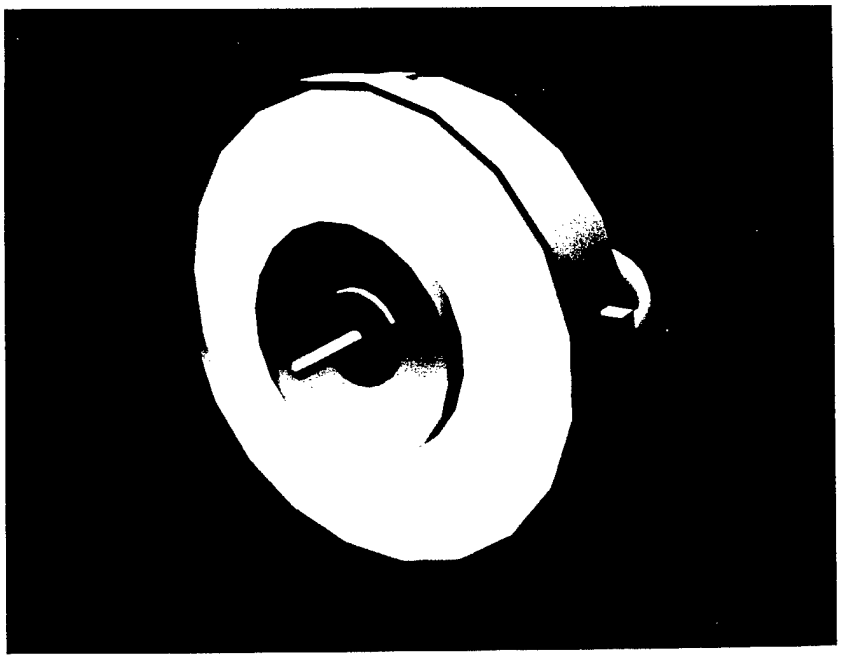
7.7 Conclusion

The Virtual Emergency Room provides a basic emergency room simulation capability. The prototype is a workable system that demonstrates the effectiveness of the design. Both the VER MSS and PCS applications are mature enough to support simulations at the procedural level. In addition, the VER provides a flexible architecture for supporting specific emergency room simulations. The VER MSS and PCS applications provide support for an open, easily adaptable communication protocol that can be changed to meet specific medical training requirements. Beyond the capabilities of the prototype, the VER provides a springboard for further medical VR research. This initial environment provides a test-bed for emergency medical simulation, and represents a first-ever capability for medical simulation using DVE technology.

Appendix A. Dynamic Geometry

This appendix presents the Performer sub-tree diagrams for all dynamic apparatus objects.

Beamlite class



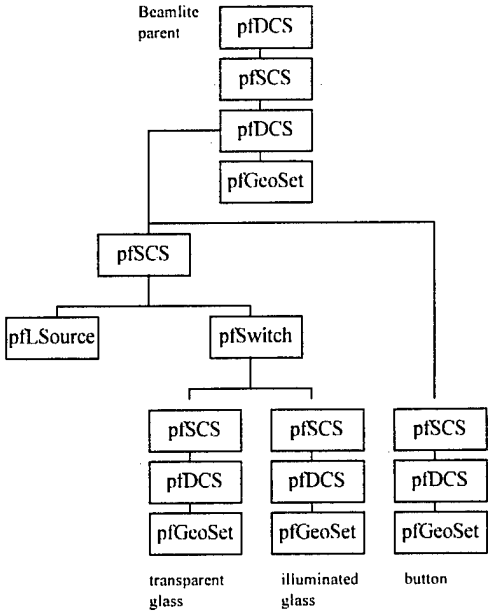
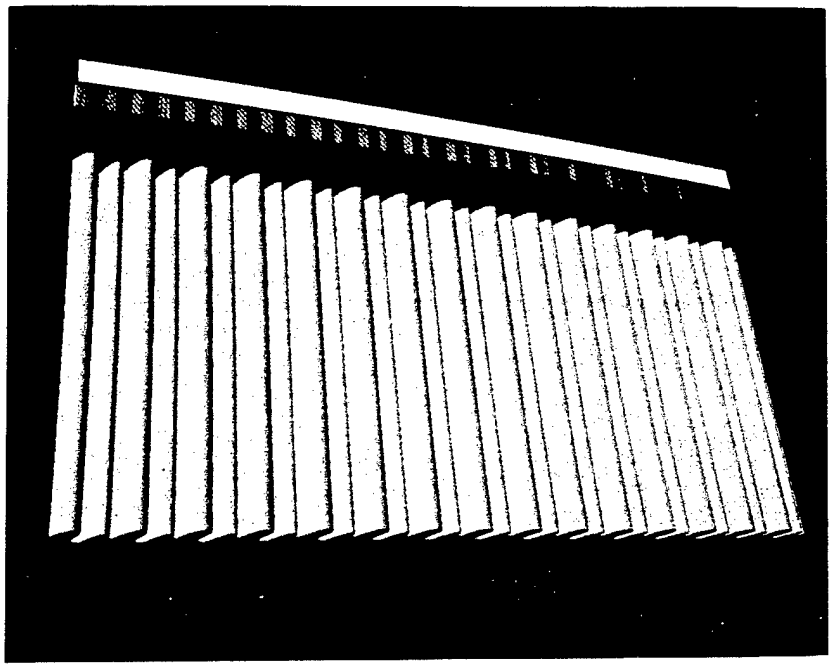


Figure A-1. Beamlite Class Performer sub-tree diagram.

Curtain class



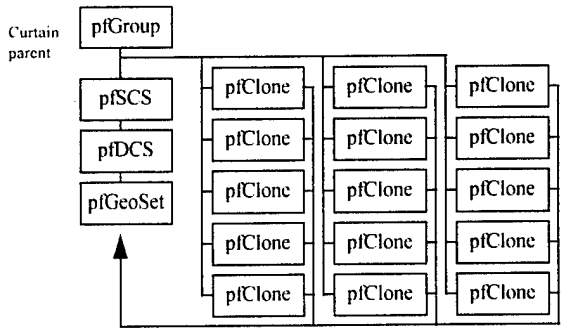
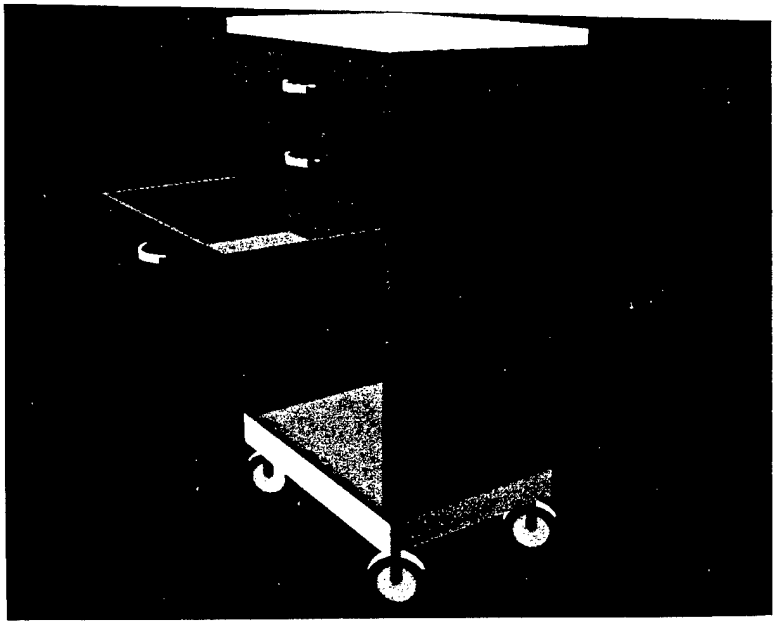


Figure A-2. Curtain Class Performer sub-tree diagram.

Crash_Cart class



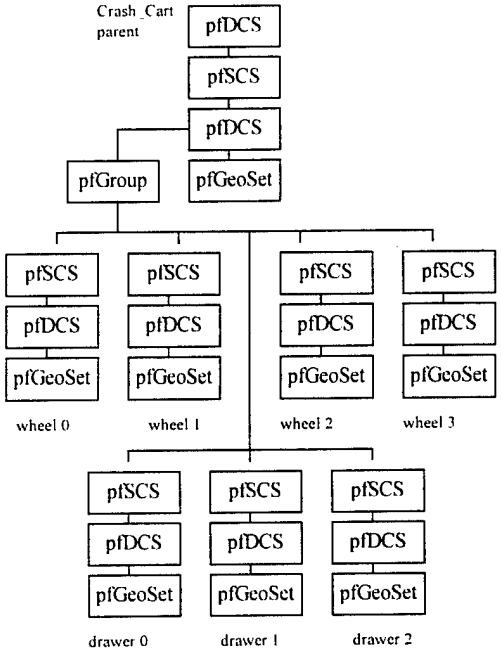
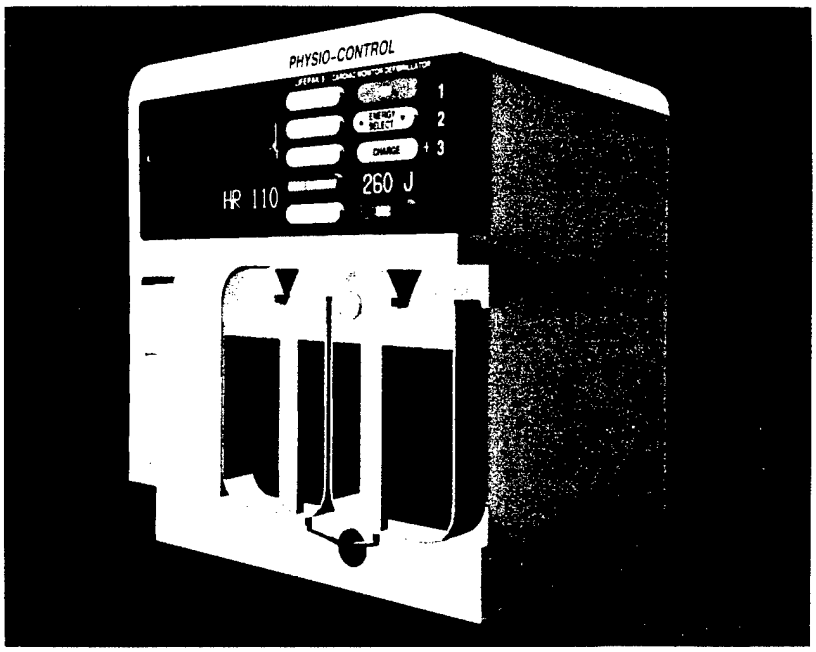


Figure A-3. Crash_Cart Class Performer sub-tree diagram.

Defibrillator class



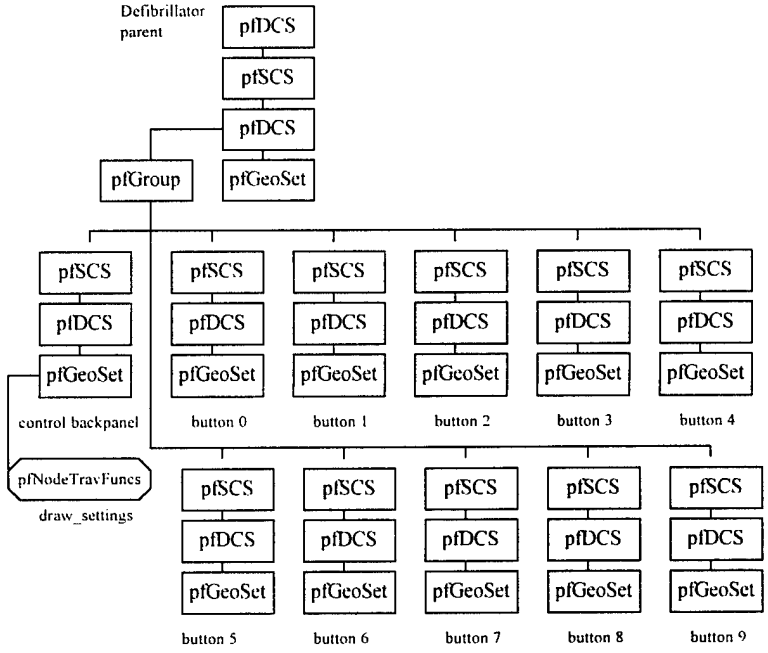
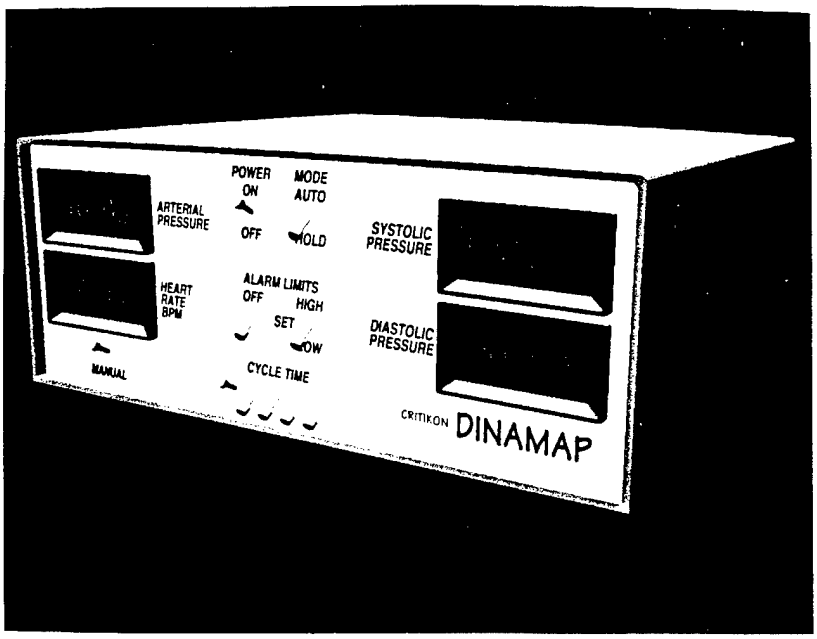


Figure A-4. Defibrillator Class Performer sub-tree diagram.

Dinamap class



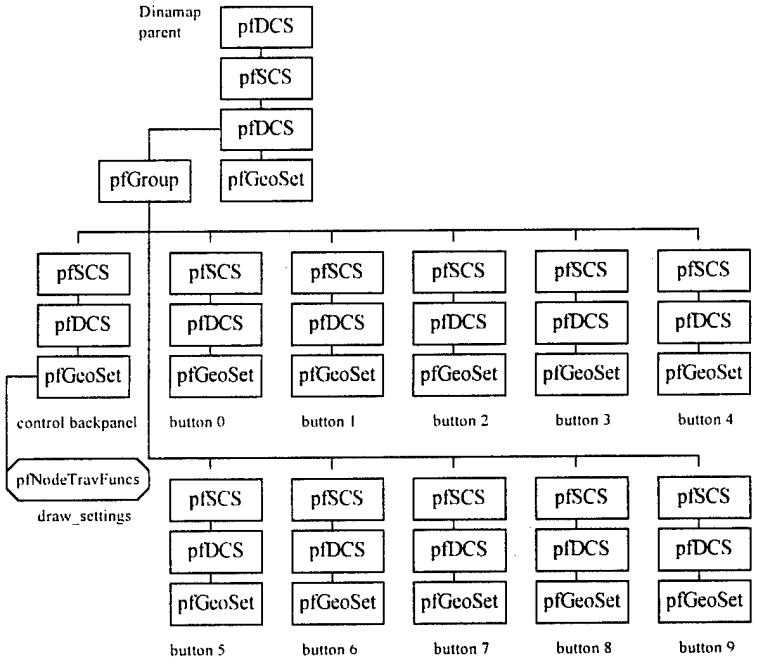
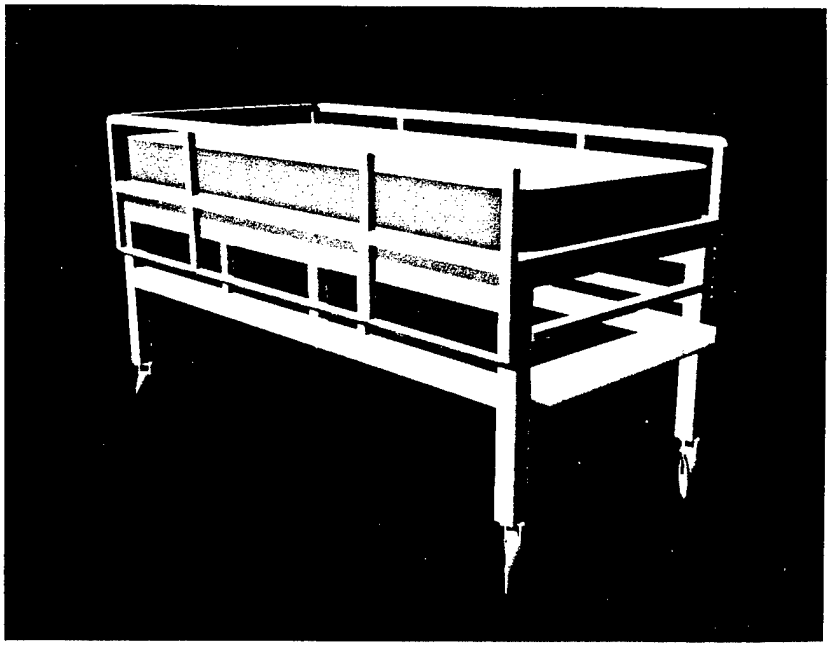


Figure A-5. Dinamap Class Performer sub-tree diagram.

Gurney class



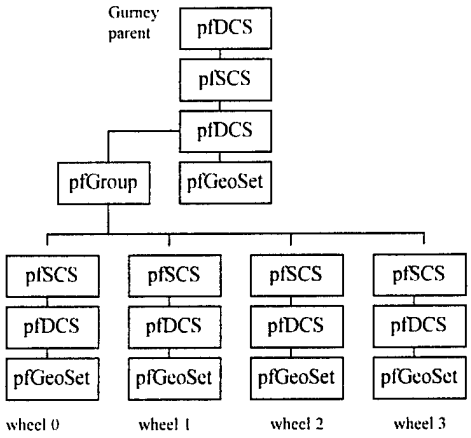
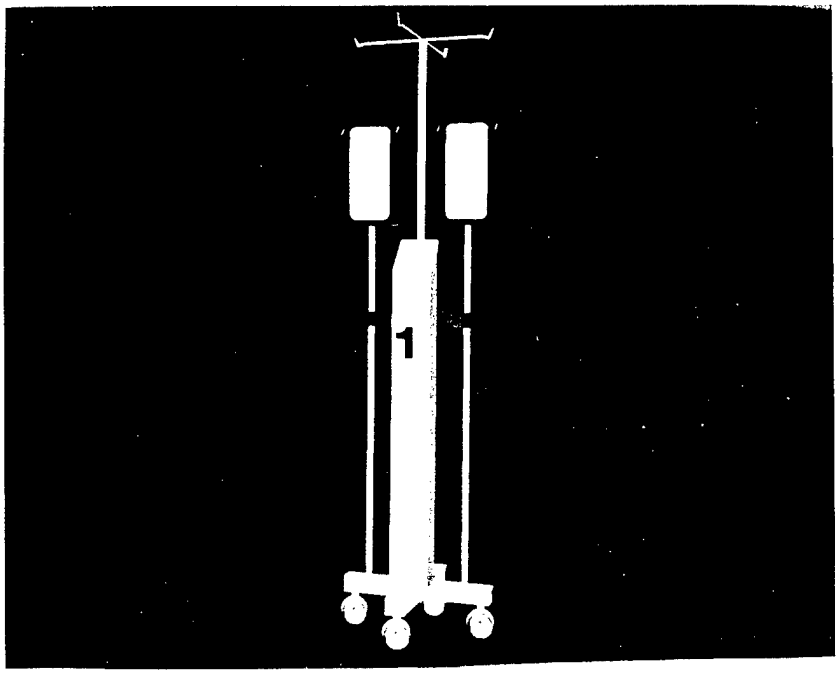


Figure A-6. Gurney Class Performer sub-tree diagram.

Infusion_Pump class



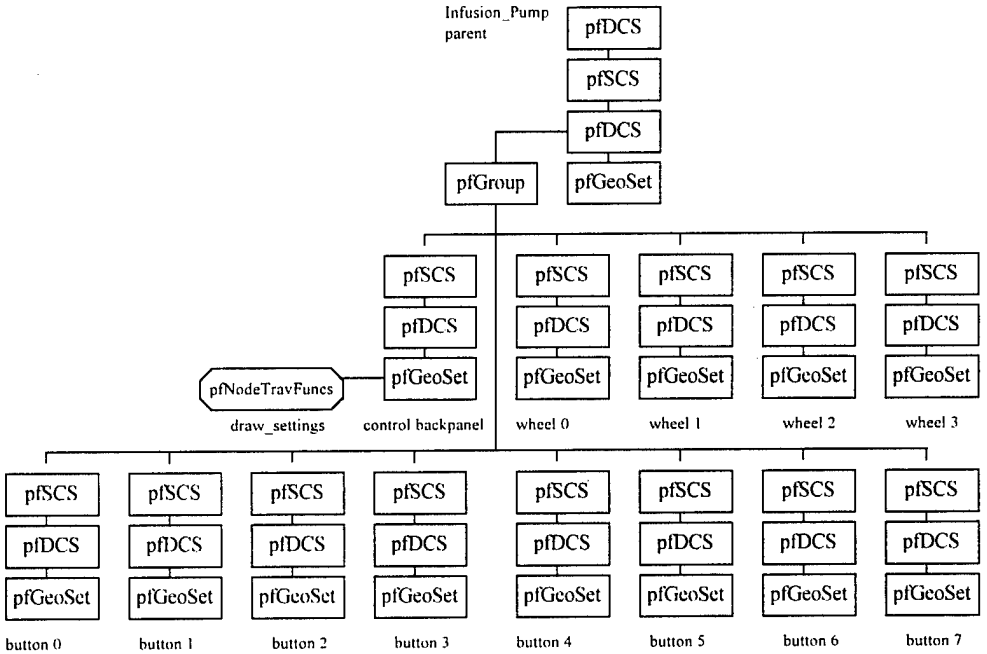
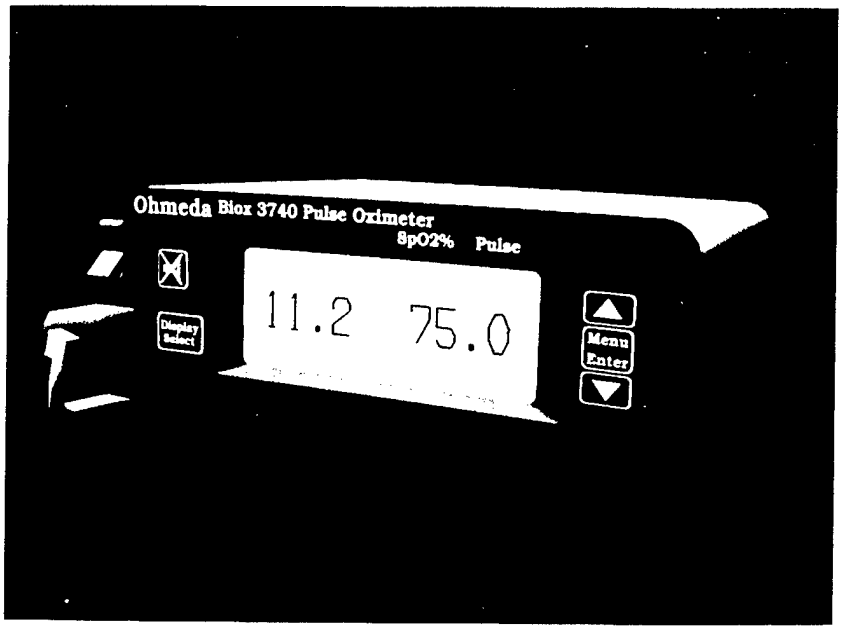


Figure A-7. Infusion_Pump Class Performer sub-tree diagram.

Oximeter class



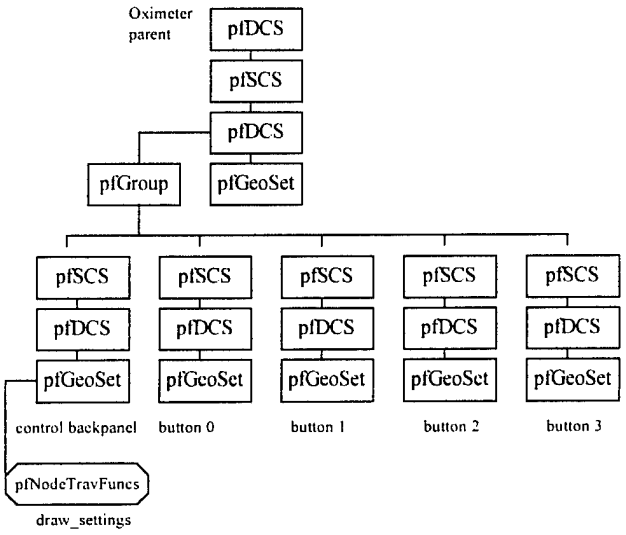
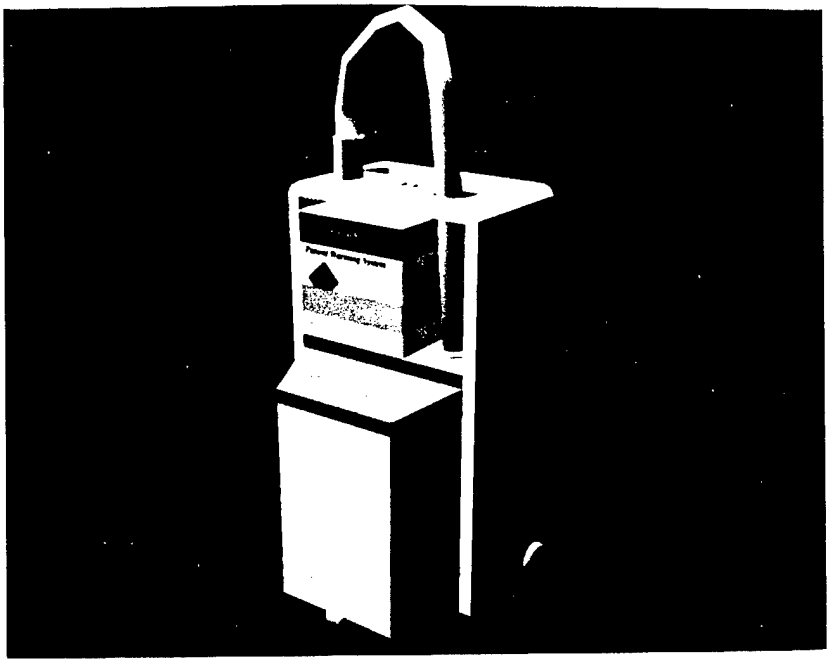


Figure A-8. Oximeter Class Performer sub-tree diagram.

Patient_Warmer class



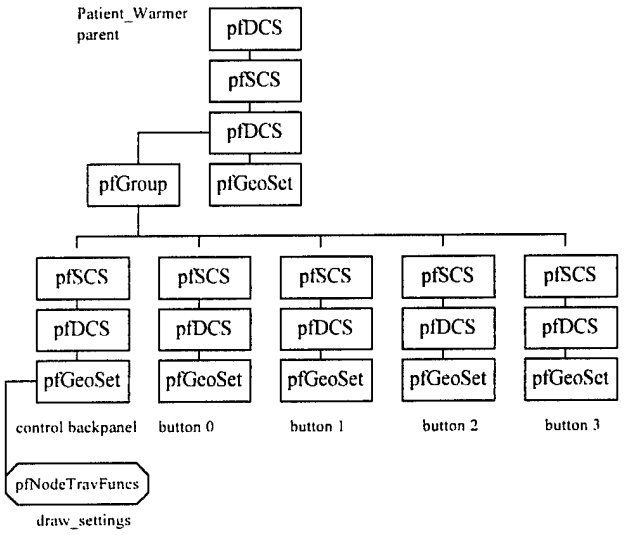
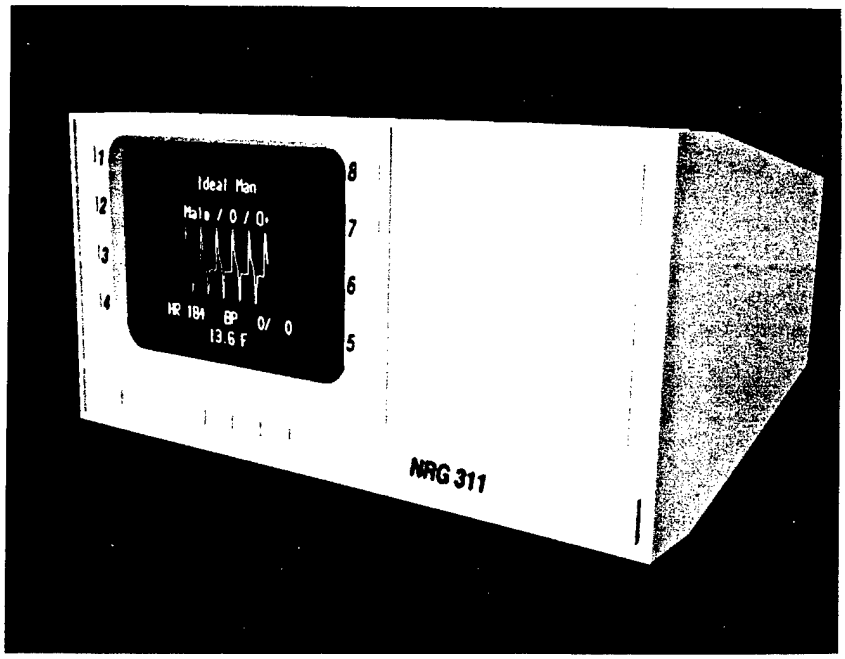


Figure A-9. Patient_Warmer Class Performer sub-tree diagram.

Primary Monitor



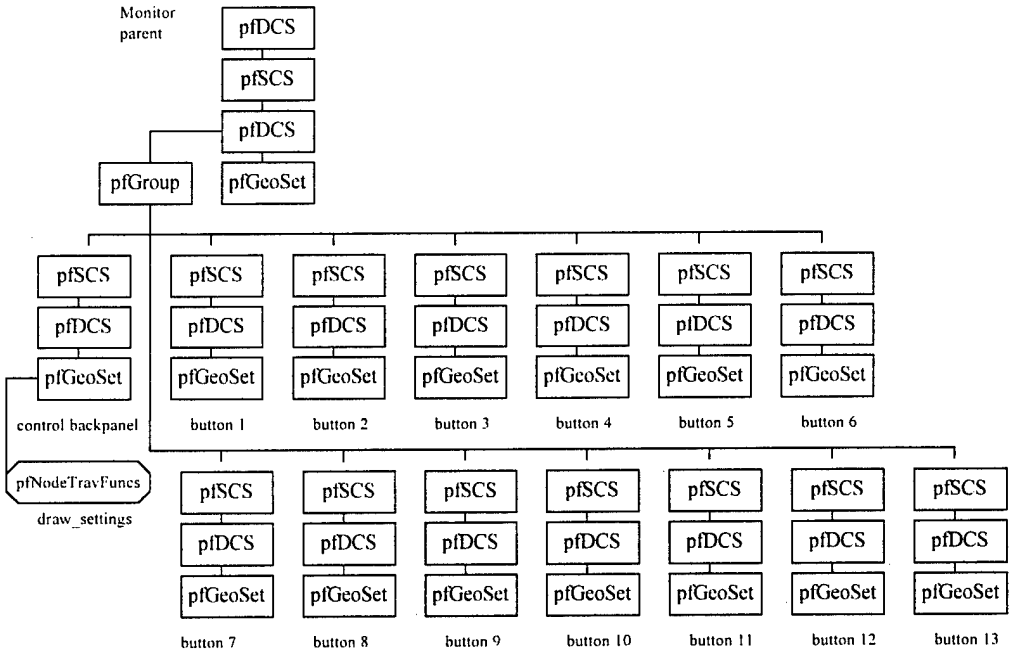
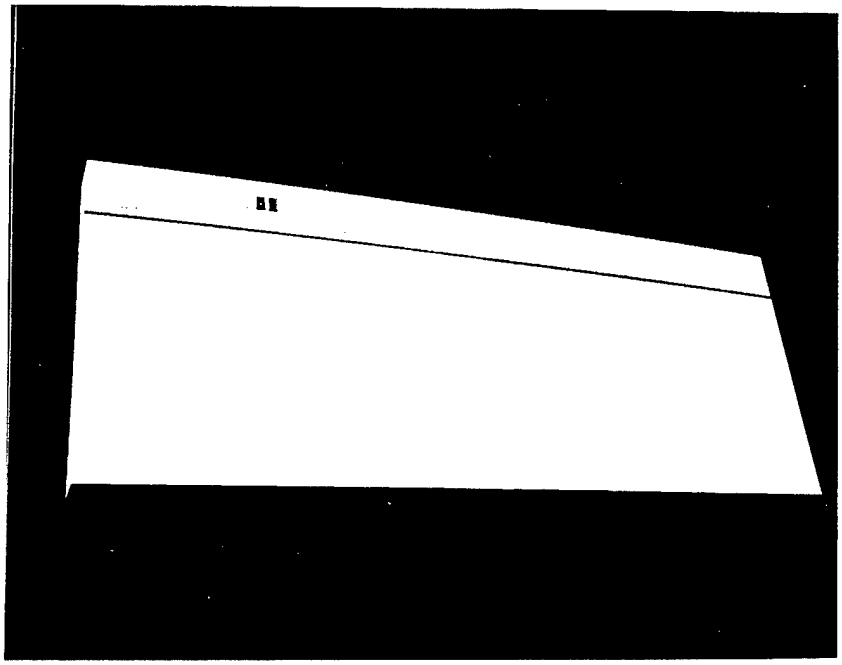


Figure A-10. Monitor Class Performer sub-tree diagram.

Xraylite class



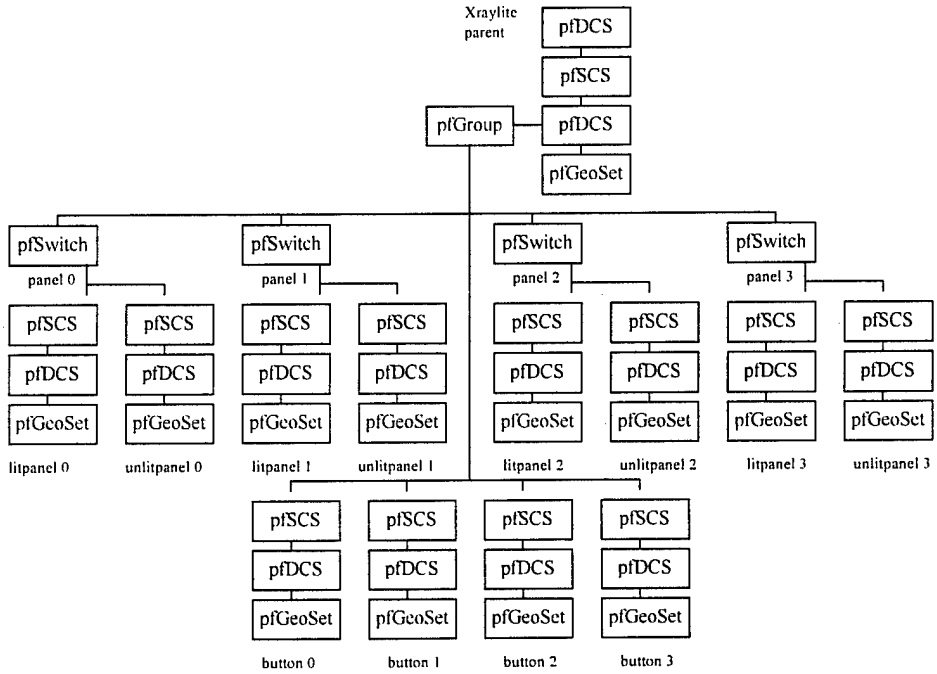


Figure A-11. Xraylite Class Performer sub-tree diagram.

Appendix B. Static Geometry

This appendix presents pictures of the static (inanimate) geometry.

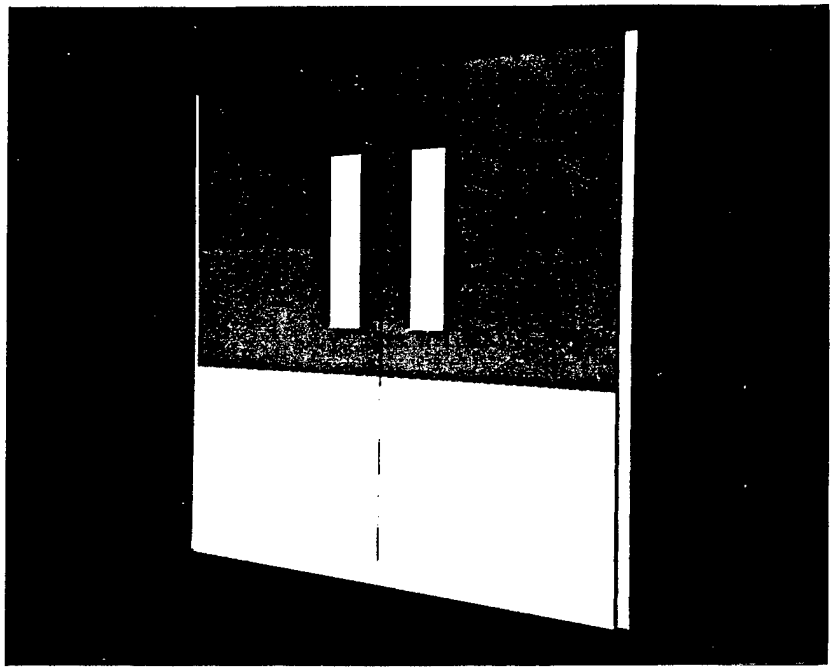


Figure B-1. Double-doors.

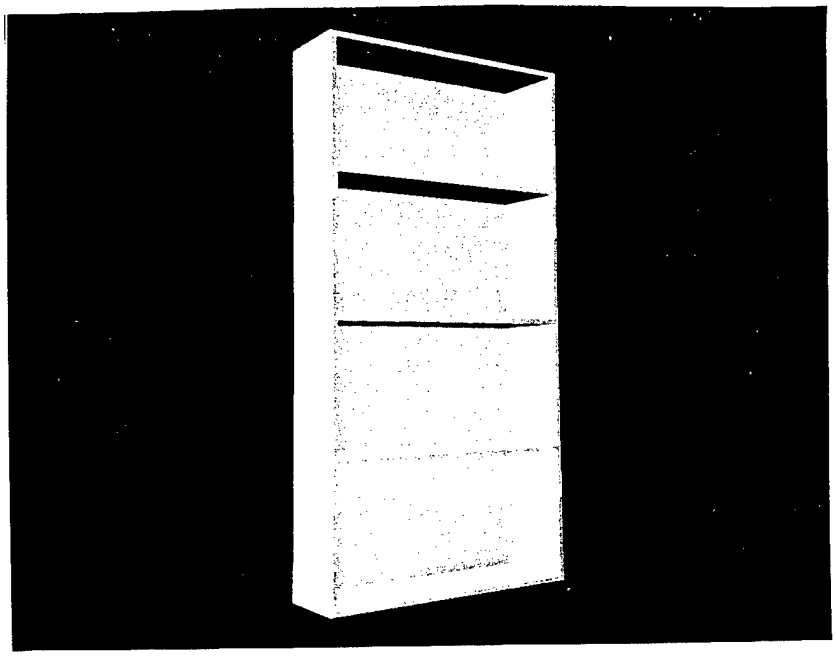


Figure B-2. Equipment cabinet.

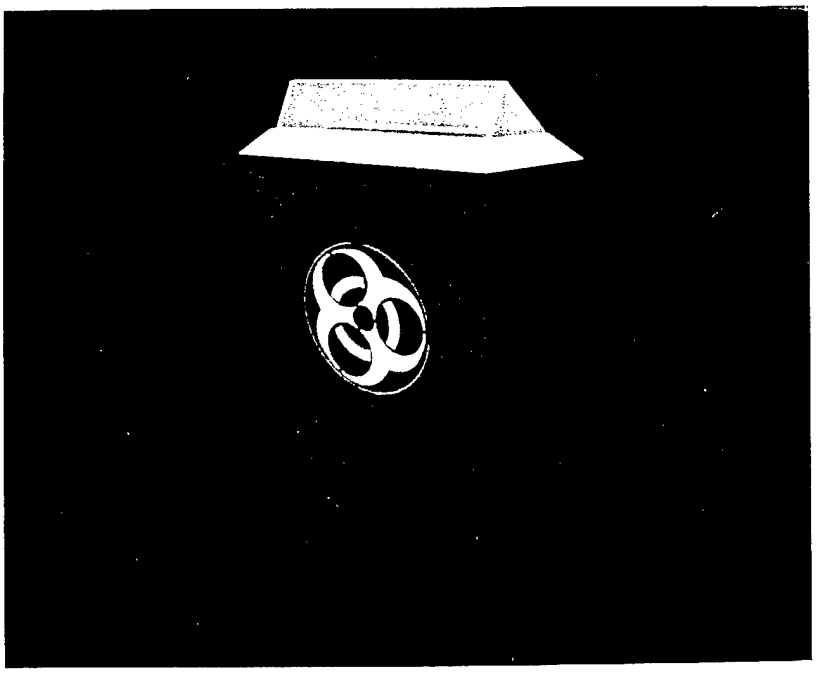


Figure B-3. Sharps waste container.

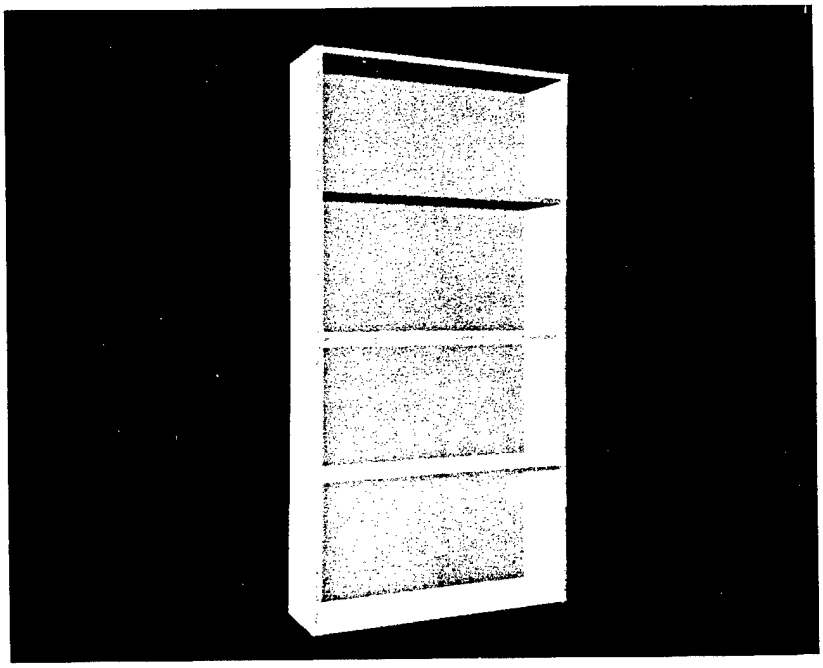


Figure B-4. Glass-covered shelves.

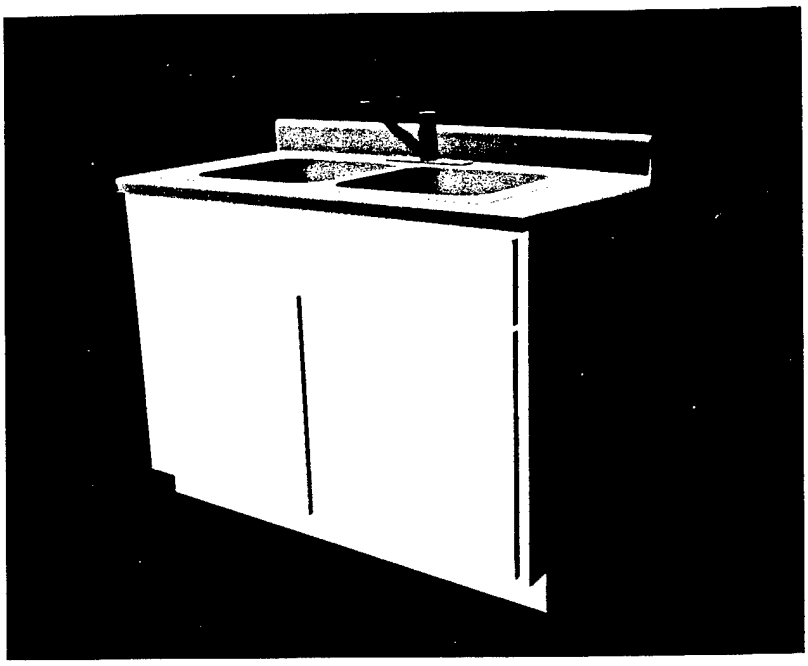


Figure B-5. Sink and cabinets.

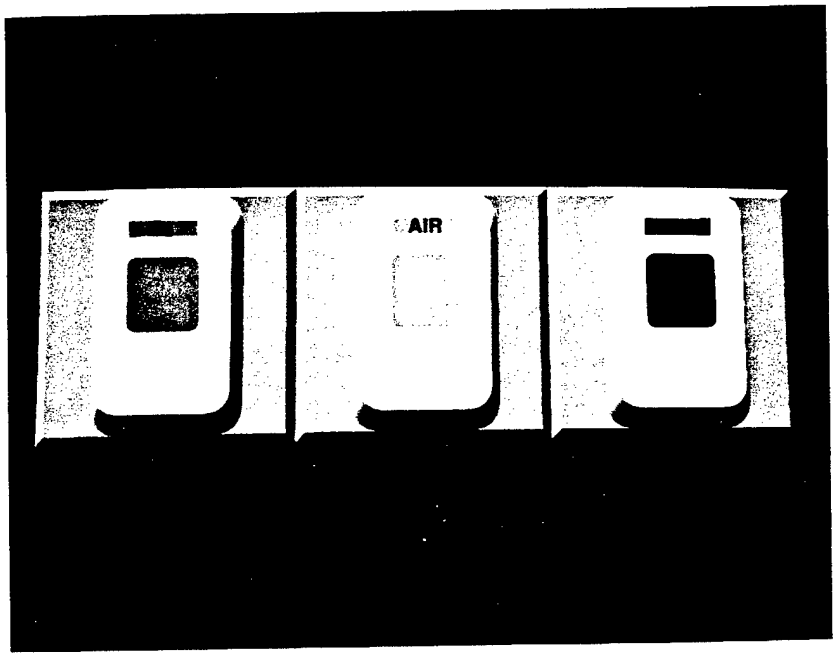


Figure B-6. Wall-mounted utilities.

Appendix C. Patient Avatar Geometry

This appendix presents the patient avatar geometry inventory adapted for the VER. The internal structure for each avatar is shown in the following Performer diagram:

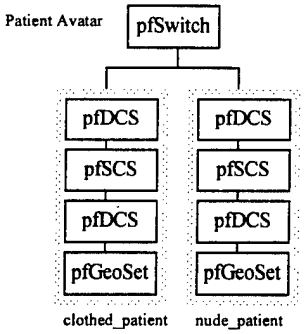


Figure C-1. Performer sub-tree diagram for patient avatar.

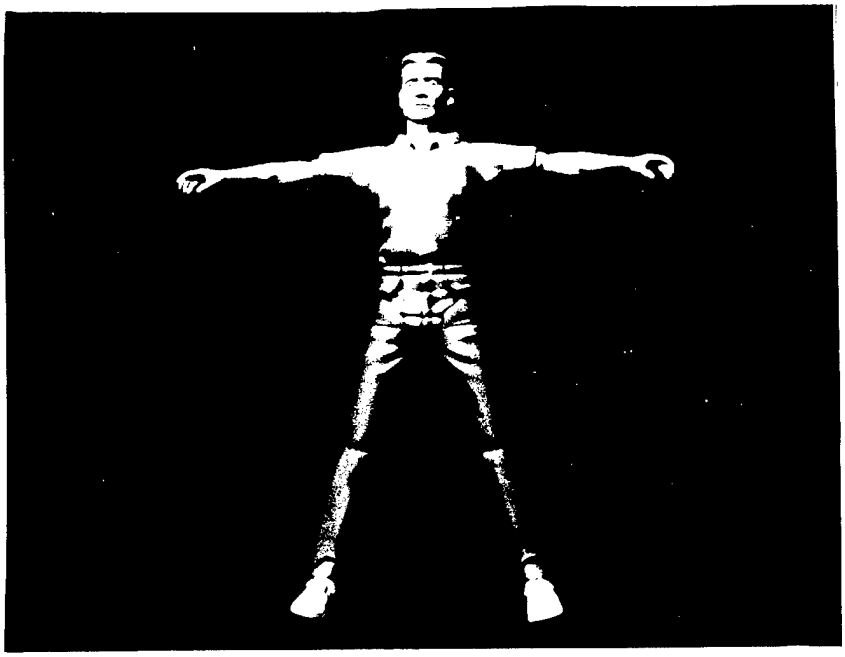


Figure C-2. Ideal Male dressed.

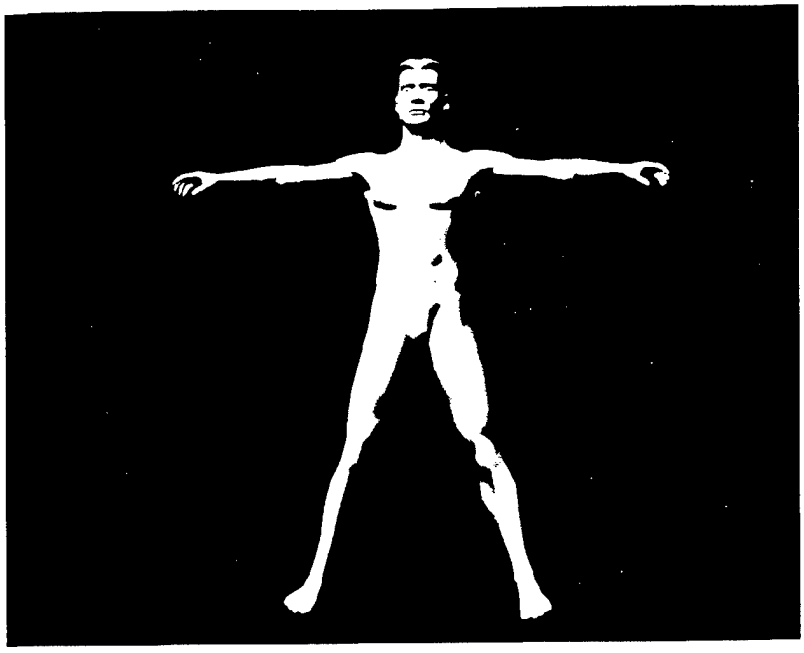


Figure C-3. Ideal Male undressed.

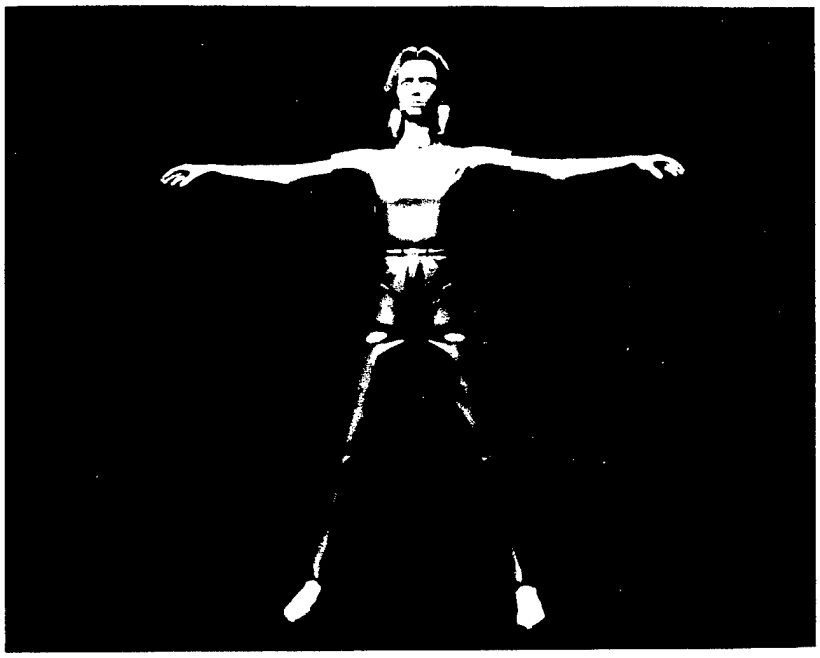


Figure C-4. Ideal Female dressed.

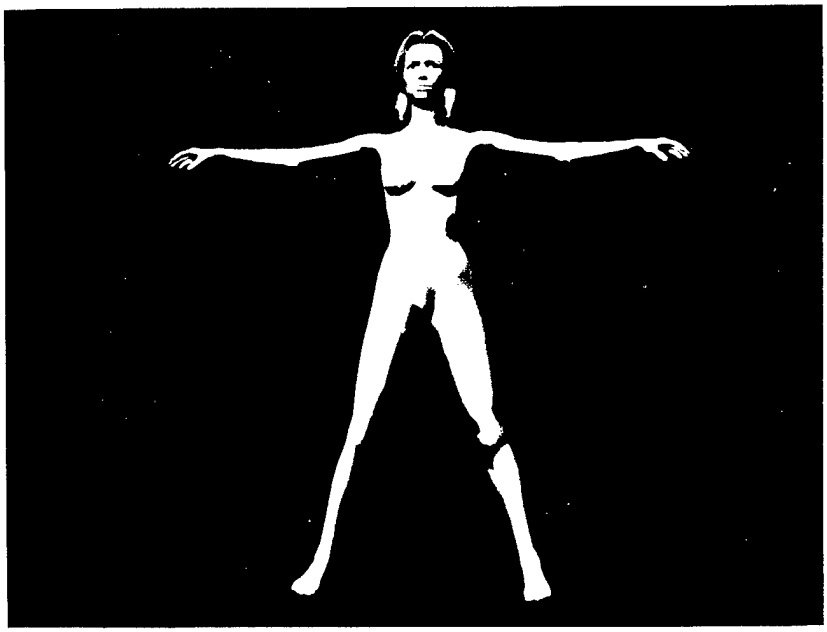


Figure C-5. Ideal Female undressed

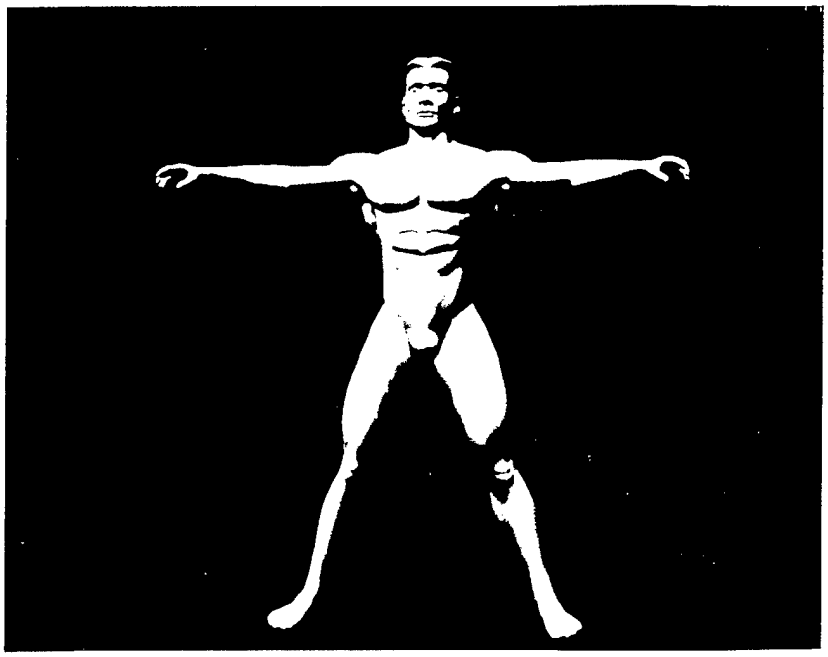


Figure C-6. Muscular Male undressed.

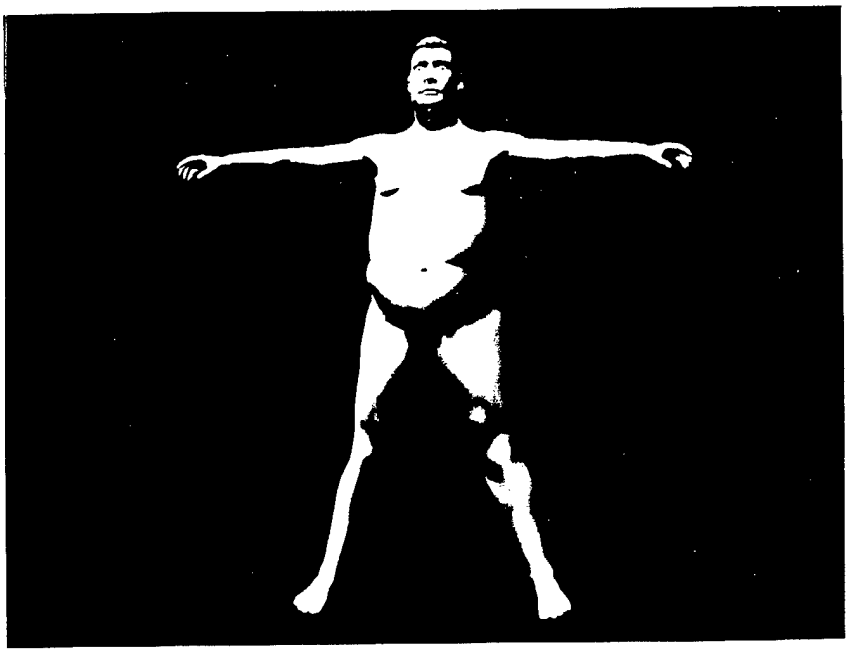


Figure C-7. Obese Male undressed.

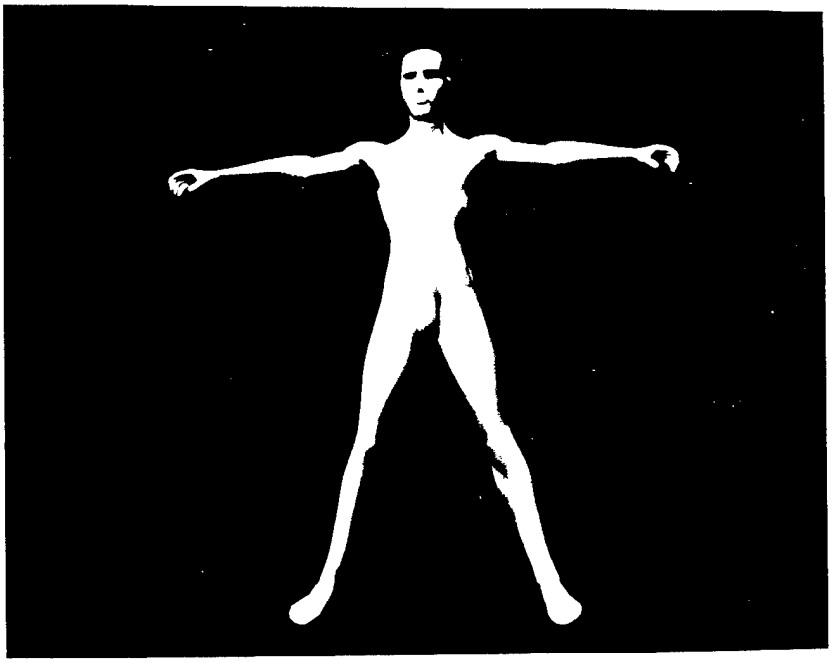


Figure C-8. Stylized Male undressed.

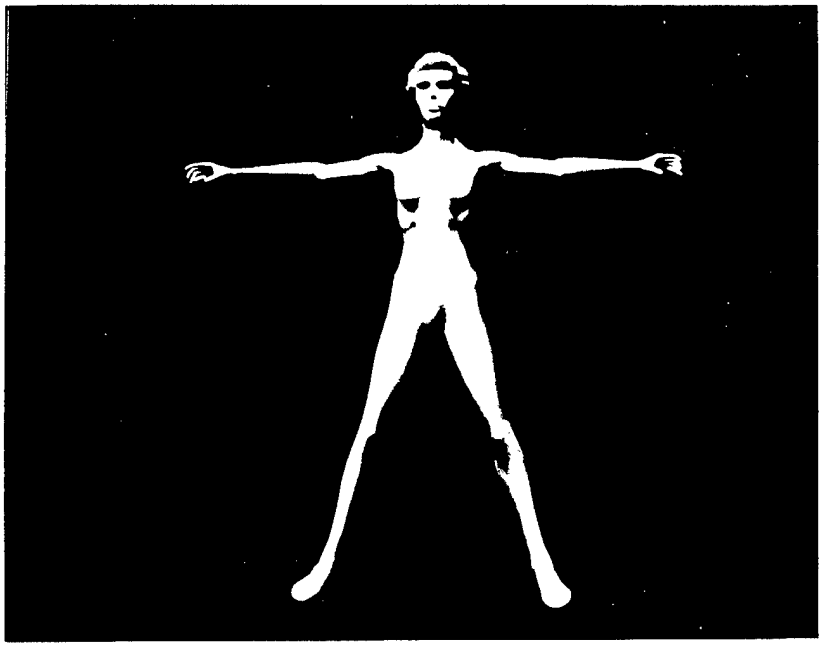


Figure C-9. Stylized Female undressed.

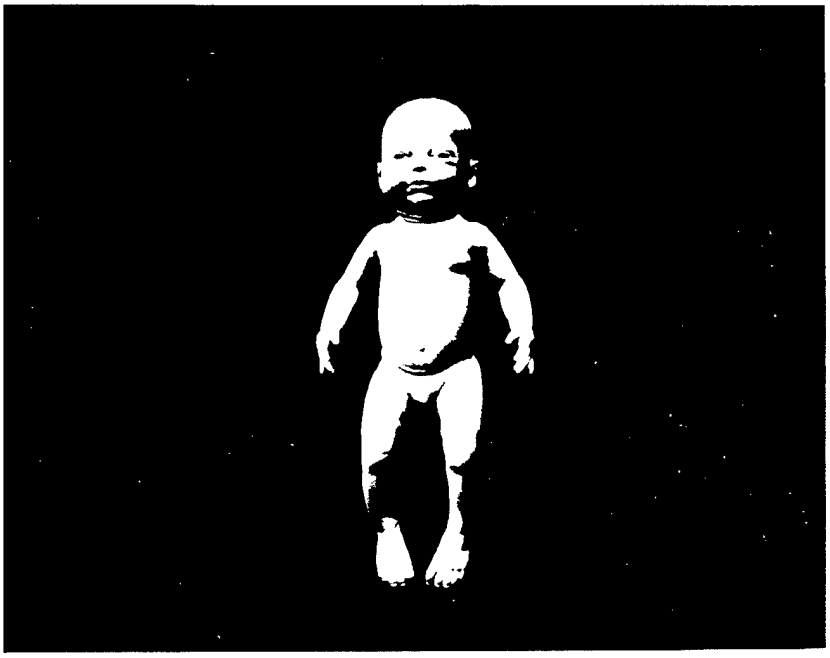


Figure C-10. Infant Male undressed.

Bibliography

ADAM96 Adams, Terry A., "Requirements, Design, and Development of a Rapidly Reconfigurable Photorealistic Virtual Cockpit Prototype," AFIT/GCS/ENG/96D-02, MS Thesis, Air Force Institute of Technology, Air University, 1996.

AHD85 The American Heritage Dictionary, 2nd College Ed., Houghton Mifflin, Boston, 1985.

BREE96 Breen, Paul T., Georges G. Grinstein, Jeffrey R. Leger, David A. Southard, and Michael A. Wingfield, "Virtual Design Prototyping Applied to Medical Facilities", in *Proceedings of the Medicine Meets Virtual Reality 4 Conference*, IOS Press, 1996, 388-399.

CGW96 Tech Watch - A Virtual Emergency Room, Computer Graphics World, Vol. 19, Number 9, 1996.

COLE94 Coleman, James, "Virtual Reality in Medicine", in *Proceedings of the fourth annual conference on Virtual Reality VR '94*, February 1994, 169-173.

CUMM94 Cummins, Richard O. Ed., Textbook of Advanced Cardiac Life Support, American Heart Association, 1994.

DMSO96 Defense Modeling and Simulation Office, *Annotated Briefing on the DoD High Level Architecture for Simulation*, September, 1996.

DUMA93 Dumay, Andrie C. M. and G. J. Jense, "On the Feasibility of Virtual Environments in Medicine," in *Proceedings of AGARD Meeting on Virtual Interfaces: Research and Applications*, October 1993, 1-8.

DUMA96 Dumay, Andrie C. M., "Triage Simulation in a Virtual Environment", in *Proceedings of Medicine Meets Virtual Reality 4 Conference*, IOS Press, 1996, 101-111.

EISL96 Eisler, R. D., A. K. Chatterjee, and G. H. Burghart, "Simulation and Modeling of Penetrating Wounds from Small Arms," in *Proceedings of the Medicine Meets Virtual Reality 4 Conference*, IOS Press, 1996, 511-522.

ELIO95 Eliot, Chris and Beverly Park Woolf, "An Adaptive Student Centered Curriculum for an Intelligent Training System", *User Modeling and User-Adapted Interaction*, Kluwer, 1995, pp. 67-86.

GODS95 Godsell-Stytz, Gayl M., and Martin R. Stytz, "The Virtual Emergency Room: A Distributed Interactive Virtual Environment for Emergency Medical Training," in *Proceedings of the 1995 Distributed Interactive Simulation (DIS) Workshop*, 1-9.

GREE95 Greenleaf, Walter J., "Virtual Reality Applications in Medicine", in *Proceedings of the Wescon '95 Conference*, 691-696.

GREE96 Greenleaf, Walter J., "Developing the Tools for Practical VR Applications", *IEEE Engineering in Medicine and Biology*, March/April, 1996, 23-30.

GUPT95 Gupta, Subnas C., Scott A. Klein, John H. Barker, Ralph J. P. M. Franken, Joseph C. Banis, Jr., "Introduction of New Technology to Clinical Practice: A Guide for Assessment of New VR Applications", *The Journal of Medicine and Virtual Reality*, Spring 1995, 16-20.

HON96 Hon, David, "Medical Reality and Virtual Reality", in *Proceedings of the Medicine Meets Virtual Reality 4 Conference*, IOS Press, 1996, 327-341.

HUAN95 Huang, Zhiyong, Ronan Boulic, Nadia Magnenat Thalmann and Daniel Thalmann, "A Multi-Sensor Approach for Grasping and 3D Interaction", Computer Graphics: Developments in Virtual Environments, Chapter 17, 1995, 235-253.

IRIS95 IRIS Performer Programmer's Guide, Silicon Graphics, Inc., 1995

IST94 Standard for Distributed Interactive Simulation: Application Protocols, Version 2.0, Draft 4, Institute for Simulation and Training, IST-CR-93-40.

JENK78 Jenkins, A. L., and John H. van de Leuv. Eds., Emergency Department Organization and Management. 2nd Ed., American College of Emergency Physicians, Mosby, St. Louis, 1978.

KAPL96 Kaplan, Kenneth, Ian Hunter, Nathaniel I. Durlach, Daniel L. Schodek, and David Rattner, "A Virtual Environment for a Surgical Room of the Future", in *Proceedings of Medicine Meets Virtual Reality 4 Conference*, IOS Press, 1996, 161-167.

KERR96 Kerr, John, Peter Ratiu, and Mike Sellberg, "Volume Rendering of Visible Human Data for an Anatomical Virtual Environment", in *Proceedings of the Medicine Meets Virtual Reality 4 Conference*, IOS Press, 1996, 352-367.

LORE95 Lorensen, B., "Marching through the Visible Man," Unpublished, http://www.ge.com/crd/ivl/vm/vm.html, GE Imaging & Visualization Laboratory, Summer 1995, 1-8.

MCGO96 McGovern, Kevin, and Rob Johnston, "The Role of Computer-Based Simulation for Training Surgeons", in *Proceedings of Medicine Meets Virtual Reality 4 Conference,* IOS Press, 1996, 342-345.

MERR94a Merril, Jonathan R., "Why I Simulate Surgery: Notes from the Surgical Field," *Virtual Reality World*, November-December 1994, 54-57.

MERR94b Merril, Jonathan R., "VR for Medical Training and Trade Show 'Fly-Paper", *Virtual Reality World*, May/June 1994, 53-57.

MOCH92 Mochizuki, Kenji, Haruo Takemura, Fumio Kishino, "Object Manipulation and Layout in a 3-D Virtual Space Using a Combination of Natural Language and Hand Positioning", in *Proceedings of the International Society for Optical Engineering*, SPIE Sensor Fusion V Conference, November 1992, 106-113.

NIEL93 Nielsen, Jakob., Usability Engineering, AP Professional, 1993. 115-227.

NILA93 Nilan, Michael S., "Task-Specific Usability Requirements for Virtual Information Environments: Interface Design and Data Representation for Human Operators of Complex Medical Systems," in *AGARD Meeting on Virtual Interfaces: Research and Applications Proceedings*, October 1993, 1-8.

NURS80 Nursing80 Photobook: Dealing with Emergencies, Intermed Communications, Inc., Horsham, 1980.

PEUC95 Peuchot, Bernard, Alain Tanguy and Michel Eude, "Virtual Reality as an Operative Tool During Scoliosis Surgery", in *Proceedings of the Computer Vision, Virtual Reality, and Robotics in Medicine: First International Conference (CVRMed-95)*, 3-6 April, 1995, 548-554.

POST96a Poston, Tim, Luis Serra, Meiyappan Solaiyappan, and Pheng Ann Heng, "The Graphics Demands of Virtual Medicine", *Computers and Graphics*, Vol. 20, Number 1, January/February 1996, 61-68.

POST96b Poston, Timothy, and Luis Serra, "Dexterous Virtual Work", *Communications of the ACM*, Vol. 39, No. 5, May 96, 37-45.

ROBB96 Robb, R. A., "VR Assisted Surgery Planning Using Patient-Specific Anatomic Models", *IEEE Engineering in Medicine and Biology*, March/April, 1996, 60-69.

ROSE96 Rosen, Joseph M., Hooman Soltanian, Donald R. Laub, Adam Mecinski, and Wendy K. Deam, "The Evolution of Virtual Reality from Surgical Training to the Development of a Simulator for Health Care Delivery, A Review.", in *Proceedings of the Medicine Meets Virtual Reality 4 Conference*, IOS Press, 1996, 89-99.

ROSE96b Rosen, Joseph M., Hooman Soltanian, Richard J. Redett and Donald R. Laub, "Evolution of Virtual Reality: From Planning to Performing Surgery", *IEEE Engineering in Medicine and Biology*, March/April, 1996, 16-22.

RUMB91 Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, Object-Oriented Modeling and Design, Prentice-Hall, Englewood Cliffs.

SATA93a Satava, Richard M., "Surgery 2000: A Technologic Framework for the Future", in *Proceedings of the Third Annual Conference on Virtual Reality*, Mecklermedia, 1993, 101-102.

SATA93b Satava, Richard M., "Virtual Reality Surgical Simulator: The First Steps," in *Proceedings of the Third Annual Conference on Virtual Reality*, 1993, Mecklermedia, 1993, 103-105.

SATA94 Satava, Richard M., "Update on Virtual Reality in Health Care: The New Future for Medicine and Surgery," in *Proceedings of the Fourth Annual Conference on Virtual Reality*, February 1994, 174-180.

SATA95 Satava, Richard M., "Virtual Reality and Telepresence for Military Medicine," *Computers in Biology and Medicine*, Vol. 25, 1995, 229-236.

SATA96a Satava, Richard M., "Medical Virtual Reality: The Current Status of the Future," in *Proceedings of the Medicine Meets Virtual Reality 4 Conference*, IOS Press, 1996, 100-106.

SATA96b Satava, Richard M. and Shaun B. Jones, "An Integrated Medical Virtual Reality Program: The Military Application," *IEEE Engineering in Medicine and Biology*, March/April 1996, 94-97+.

SCHW89 Schwartz, George R., Nicholas Bircher, Barbara K. Hanke, Mary Anne Mangelsen, Thom Mayer, and James R. Ungar, Emergency Medicine: The Essential Update, W. B. Saunders, Philadelphia, 1989.

SELL95 Sellberg, Michael, Daniel Murray, Darren Knapp, Todd Teske, Katherine Lattie, and Martin Vanderploeg, "Virtual Human", in *Proceedings of the Medicine Meets Virtual Reality 3 Conference*, IOS Press and Ohmsha, 1995, 340-347.

SHEA96 Sheasby, Steve, Technical collaboration. September-October, 1996.

SHEE92 Sheehy, Susan B., Emergency Nursing Principles and Practice, 3rd Ed., Mosby, St. Louis, 1992.

STYT97 Stytz, Martin R., Terry Adams, Brian Garcia, Steven S. Sheasby, and Brian Zurita, "Developments in Rapid Prototyping and Software Architecture for Distributed Virtual Environments," To Appear in *IEEE Software*, 1997, 1-30.

THAL94 Thalmann, Nadia Magnenat and Daniel Thalmann, "Towards Virtual humans in Medicine: A Prospective View," *Computerized Medical Imaging and Graphics*, Vol. 18, 1994, 97-106.

VINC95 Vince, John, Virtual Reality Systems, Addison-Wesley, 1995, 301-303.

WELL96 Wells, William D., "Collaborative Workspaces Within Distributive Virtual Environments", MS Thesis, Air Force Institute of Technology, Air University, 1996.

WILL96 Williams, Gary E., "Solar System Modeler: A Distributed, Virtual Environment for Space Visualization and GPS Navigation", AFIT/GCS/ENG/96D-29, MS Thesis, Air Force Institute of Technology, Air University, 1996.

ZHOU96 Zhao, T. C. and Mark Overmars, Forms Library: A Graphical User Interface Toolkit for X, V0.81, July 1996.

Vita

Capt Brian W. Garcia was born on 6 August 1968 in El Paso, TX. In 1986 he accepted an appointment to the United States Air Force Academy to earn a Bachelor's degree in Computer Science. Upon graduation and commissioning with the Class of 1990, Capt Garcia attended the Basic Communications-Computer Officer Training (BCOT) course at Keesler AFB, MS. His first permanent assignment was to the Air Force Military Personnel Center (now the Air Force Personnel Center) at Randolph AFB, TX. During his four and a half year tour at Randolph AFB, Capt Garcia earned a Masters of Science in Computer Information Systems from St. Mary's University, where he was a Distinguished Graduate. In the winter of 1995, Capt Garcia attended Squadron Officer School (SOS) in residence at Maxwell AFB, AL. Shortly thereafter, Capt Garcia entered the AFIT Software Engineering program in the summer of 1995. His next assignment is to ACC CSS, Langley AFB, VA. Capt Garcia is married to the former Claudia Carrasco of El Paso, TX. He and his wife are expecting their first child in April of 1997.

Forwarding Address: 9 West Pinto Court
Hampton, VA 23666

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	3 Dec 96	Master's Thesis

4. TITLE AND SUBTITLE
DESIGN AND PROTOTYPE OF THE AFIT VIRTUAL EMERGENCY ROOM: A DISTRIBUTED VIRTUAL ENVIRONMENT FOR EMERGENCY MEDICAL SIMULATION

5. FUNDING NUMBERS

6. AUTHOR(S)
Brian W. Garcia, Captain USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology,
WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/96D-07

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Due to the increasing complexity of emergency medical care, medical staffs require increasingly sophisticated training systems. Virtual environments offer a low cost means to achieve a widely usable yet sophisticated training capability. The Defense Advanced Research Projects Agency (DARPA) has sponsored the Virtual Emergency Room (VER) project to develop a simulation system that enables emergency department personnel within level I and II emergency rooms to practice emergency medical procedures and protocols. The VER is a simulation facility that uses a distributed virtual environment architecture to enable real-time, multi-participant simulations. The potential advantages of this system include the ability to evaluate and refine treatment skills, and the ability to provide scenario-specific training for mobile military field hospital teams. These advantages will ultimately improve the readiness of emergency department staffs for a wide variety of trauma situations. This thesis represents the initial phase of a several-year research effort.

14. SUBJECT TERMS
Distributed Virtual Environments, Emergency Medical Simulation, Virtual Reality, Emergency Room, Medicine

15. NUMBER OF PAGES
192

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL